

PASCAL

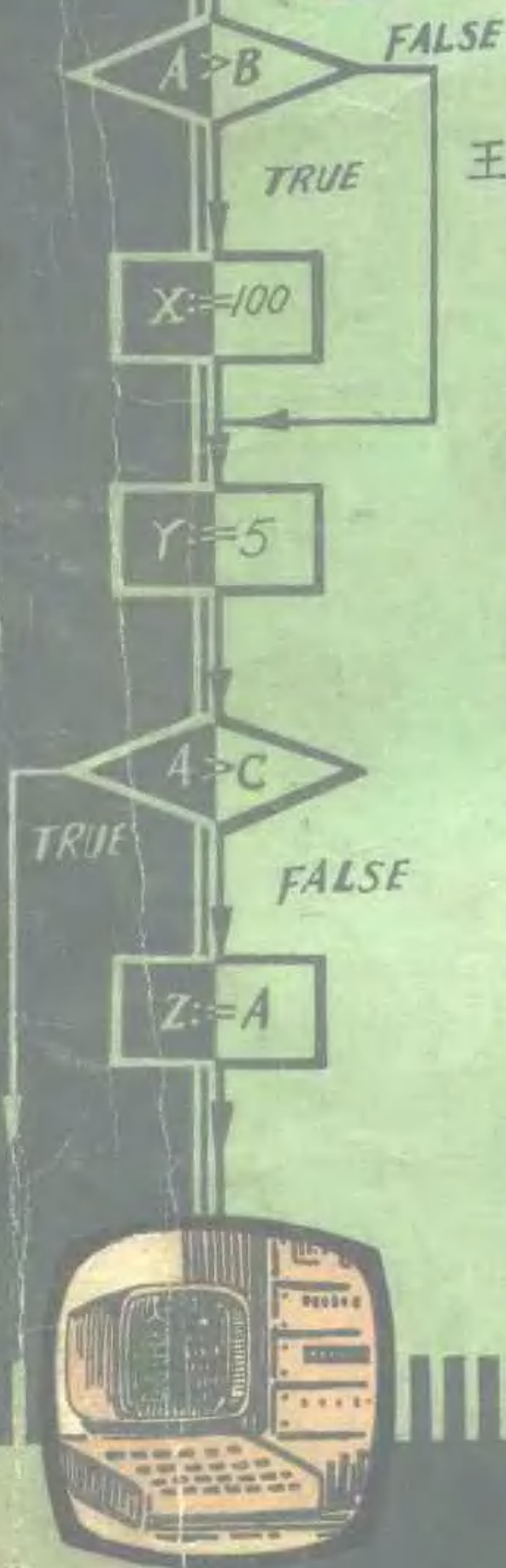
程序设计及其应用

王 诚 苏云清 赵毓陞

编 著

奚和泉 陈圣信

校 订



清华大学出版社

125

PASCAL

程序设计及其应用

王 诚 苏云清 赵毓升 编著

姜和泉 陈圣信 校订

(IS/01/33)

IS 01 33

清华大学出版社



内 容 简 介

本书是清华大学计算机工程与科学系、科学研究处在最近几年教学和科研工作的实践基础上编写的。

PASCAL 语言具有丰富完备的数据类型, 简洁灵活的通用语句, 清晰明了的模块结构, 以及编译紧凑方便, 书写格式自由, 运行效率较高和程序设计风度优美等特点。它在我国正在迅速推广, 并且日益得到重视。

本书全面、系统地介绍了PASCAL语言, 它的程序设计方法及其应用。共分十一章, 内容包括PASCAL语言的全部数据类型, 多种语句, 程序设计的基本方法、技巧和实际应用。共有各种类型的程序182个。从第一章到第九章都附有思考题和习题。通俗易懂, 深入浅出, 内容丰富, 是一本较好的学习和掌握PASCAL程序设计及其应用的教科书, 可供高等院校计算机、自动化、无线电、经济管理等有关专业师生以及其它部门的计算机工作者, 科研、工程技术人员学习参考。

PASCAL 程序设计及其应用

王 诚 苏云清 赵毓升 编著
奚和泉 陈圣信 校订

☆

清华大学出版社出版

北京 海淀 清华园

轻工业出版社印刷厂 排版

湖北省固安县印刷厂 印装

新华书店北京发行所发行 各地新华书店经售

☆

开本: 787×1092 1/16 印张: 30.5 字数: 780千字

1983年9月第一版 1983年9月第一次印刷

印数: 1—50,000

统一书号: 15235-58 定价: 3.40元

序

《PASCAL程序设计及其应用》出版了。在通读了它的全稿以后，我高兴地向读者推荐这本通俗易懂、内容丰富的科技读物。

我希望这本书能够担当起普及和提高的双重任务：一方面它是一本合用的大学教科书，另一方面它又是一本面向广大科技工作者的普及读物。

首先，PASCAL语言本身就具有“雅俗共赏”的特性。它既受到欧、美许多国家的共同重视，尤其得到高等学校计算机工作者的普遍接受，可以说是目前国际上最受欢迎、最广泛流行的程序设计语言之一。它小巧、简洁、连贯、精致，结构性好，表达能力强，实现效能高，移植容易，流传广泛，所有这些优点使PASCAL语言不但具有学院式语言的资格，而且即将成为一种新的标准化工业语言。从它的普及性来看，PASCAL语言足以和当前公认两种普及性语言BASIC和APL相比美。它象BASIC一样易于学习和使用，但在支持结构化程序设计和表达算法的能力上比BASIC优越得多。它与APL同样有效地表达算法，但具有更好的易读性。我们完全有理由通过推荐这本书向广大计算机科技工作者推荐PASCAL这样一种“前程似锦”的高级语言。

为了在我国普及计算机教育和推广计算机应用，我们迫切需要能适合我国读者情况的PASCAL方面的书籍。我们是从1979年开始较多地使用PASCAL语言的，短短几年的教学和科研实践证明它确是一种很好的计算机高级语言。如果说由于受历史原因的局限我们曾经在高等学校中不得不广泛采纳BASIC语言的话，那么今天应该创造条件逐步用PASCAL来取代BASIC语言在我国高等学校中目前占据的不适当地位。我们建议把PASCAL语言作为大学推广计算机应用的启蒙语言，让学生从一开始就培养严谨的、良好的程序设计风格和习惯，然后再学习FORTRAN、COBOL等带有一定应用偏向的语言。这样做将对学生的程序设计基本训练产生深远的影响。

我相信，这本书足以胜任上述任务。它写得概念清楚、层次分明、叙述流畅和通俗易懂。本书的作者是几位中年教师，几年来他们一直深入教学和应用第一线，积累了丰富的实践经验，又认真地学习和消化了国外有关的书籍和资料，经过多次上机调试、课堂讲授和修改校订，较好地做到了理论和实践相结合。当然，由于PASCAL语言目前正处于标准化的过程中，在我国普及还有许多工作要做，因此，本书今后还需不断修订和完善，使之更加切合不同PASCAL用户的需要。

利用这本书出版的机会，写了上面的一些话，作为向本书作者的祝贺和希望。

金 兰

1982年3月于清华大学

前 言

PASCAL 是一种较新的高级语言,是根据结构程序设计原则研究出来的。它具有丰富完备的数据类型,简明灵活的通用语句,清晰明了的模块结构,以及编译紧凑方便,书写格式自由,运行效率高和程序设计风度优美等特点。PASCAL 语言适用于教学、科学计算、管理和编写各种系统软件等。

1971年,瑞士沃斯(N·Wirth)教授正式发表 PASCAL 程序设计语言,这引起了各国学者的广泛重视与反响。十余年来的实践证明,PASCAL 语言是成功的,有生命力的。现在,它已成为国际上最广泛应用的主要算法语言之一,并且日益得到重视,可以说是前程似锦。

我国开始研究和使用的 PASCAL 语言,时间较晚。近几年来,虽然翻译出版了一些有关的资料和书籍,但由于各种条件的限制,PASCAL 语言的广泛应用还刚刚开始。随着我国四个现代化建设的发展和国内外科学技术交流的扩大,PASCAL 语言一定会在我国引起更大的注意,在计算机的发展与应用中发挥越来越大的作用。正是在这种形势下,人们迫切地需要一本通俗易懂而又比较详细的,适合我国国情的有关 PASCAL 语言方面的书籍。本书正是为了适应这种形势和要求而编写的。

从我国的实际情况出发,根据大多数初学者的特点,本书首先简明地介绍 PASCAL 语言的程序结构,数据类型和通用语句,给读者一些概貌性知识,然后逐步展开,进行比较详细的讨论。在讨论过程中,较多地采用从具体例题入手,逐步引出概念,阐述理论和作出结论的方法。在讲解具体程序时,既注意讲清编写该程序的目的和具体思路,又指出初学者易犯的一些错误,力求培养正确地进行 PASCAL 语言程序设计的良好习惯。初学者学完前三章,即可独立地编写简单然而完整的小程序。有条件的应该一边学习,一边上机实践,这对提高学习效果极为有利。按照循序渐进的原则,以后各章陆续加进新的内容,使读者既温故而又能知新,注意培养综合应用和解决实际问题的能力。

PASCAL 语言是一种比较好的系统程序设计语言。本书中介绍了用 PASCAL 语言编写操作系统和编译程序的思想,并给出一些例题或者实用程序。还介绍了在工程计算、事务管理以及计算机逻辑化简等方面的综合应用程序,这些内容难度较大,主要供有关专业人员参考。某些章节不作基本要求,书中打上“•”号作标记。

本书编写过程中,总是以标准 PASCAL 语言的有关规定为依据。但所有程序都是用 OMSI PASCAL-1 语言编写的,并在 PDP-11/03 计算机系统上进行了调试与运行。这是作者条件所决定的,但并不影响本书的通用性。本书的一个重要特点,是在教学过程中与上机实践基础上编写的,总结了在程序编写、调试与运行中遇到的一些问题和实际体会。这对初学者是有益的。

由于在不同的计算机系统中运行的 PASCAL 语言存在一些差异,不同的语言版本对程序的编制和运行也有自己的某些特点,读者在使用时必须注意,要参考有关说明书中的具体规定。本书编写程序所用的 OMSI PASCAL-1 语言版本,是它的第一版的修改版,改

进后的 OMSI PASCAL-1 V1.2 版本，有了一点小改动*。请读者注意二者的区别。

在编写本书的过程中，得到了金 兰教授的热心指导和帮助，他认真地审阅了全书，提出了宝贵的意见，并为本书写了“序”。单 娟同志对本书的内容作了仔细审查，提出了修改意见。计算机实验室的同志们，在本书的编写和校订过程中，自始至终给予了良好的协助和配合。另外，我们还得到了兄弟院校和其它部门的计算机工作者的热情鼓励和帮助，在此，我们一并表示衷心的感谢。

由于我们水平有限，又缺乏经验，加上时间又比较局促，本书一定有不少缺点和错误，敬请有关老师、计算机工作者和广大读者批评指正。

编者

-
- OMSI PASCAL-1 V1.2 对 V1.1 作了如下修改，
 1. 交互终端文件中的 EOLN () 的初值由“真”改为“假”；
 2. READ () 过程，要读入开头部分的空格字符，以行结束符或终端字符数组结束输入；
 3. 文本文件类型必须用 TEXT 予以说明，而不能用 FILE OF CHAR 加以说明；
 4. 用新给出的 SEEK () 过程替代 V1.1 中的 SEEK ()、DEPOSIT () 和 CLOSERANDOMFILE () 等三个过程。

阅 读 须 知

本书内容大致分为三部分。

前五章为第一部分。这五章介绍了 PASCAL 语言的基本部分,包括 PASCAL 语言的程序结构,标准数据类型,各种通用语句以及过程和函数的应用。请读者比较与其它算法语言的差别,注意在流程控制、过程和函数方面 PASCAL 语言的特点。

第六章到第九章为第二部分。这四章介绍了 PASCAL 语言在数据类型定义方面的特点。这是本书的重点部分,也是 PASCAL 语言的精华所在。请注意,构造型数据类型——集合类型、记录类型、数组类型和文件类型,动态数据类型——指针的引进和展开,充分显示了 PASCAL 语言的优点。

第十章到第十一章为第三部分。它介绍了 PASCAL 语言源程序的运行和调试,以及如何应用 PASCAL 语言解决实际问题。运行和调试是实践的基础内容,有条件上机的读者应该提前阅读这一章。综合应用程序举例一章,部分程序难度较大,涉及的专业较多,读者可以根据实际情况进行选择。所举例题也仅供参考。

本书的实践性较强,例题很多。这有助于读者从模仿开始,逐步掌握到举一反三、灵活应用。这些程序大致分为三类,读者可以自行挑选。

第一类,程序简短。它是为了反复说明一些基本概念,从第三章开始,几乎每一节都有这一类程序。

第二类,程序稍长,层次稍复杂一些。它是为综合应用某一章的基本概念而编写的。每一章后面的“应用举例”,就是这一类程序。

第三类,程序较长,结构比较复杂。这些程序是从多方面实际应用中提取出来的,有很大实用性。选取的目的是为培养综合应用 PASCAL 语言的各种基本概念解决实际问题的能力。本书最后一章的应用举例属于这一类程序。

由于使用算法语言的同志基本上对计算机及程序设计有所了解,本书没有详细叙述这些基础知识,只是在第一章里简要地提示一下。

本书中提出的一些思考题,主要是为了巩固对基本概念的理解,提高对出现的各种问题的分析能力。

在各章后边,一般都有本章小结。小结的综合性较强,有助于读者分析和比较本章各节的内容,提高概括和综合能力。各章的例题或习题,便于读者亲自实践,以帮助掌握基本知识并用于解决实际问题。

本书中的标识符,贯彻“常用从简”的原则,多数用一二个字符表示,专用的标识符,应考虑其主要特点,用英文或英文缩写表示。例证程序,一般按章节取名,按例题编号,按字母分挡。如 SMP01, CTL02, PF03, ARY04, SET05, USE06, RCD07, FIL08, PNT09等分别表示简单程序 (SIMPLE), 流程控制 (CONTROL), 过程和函数 (PROCEDURE AND FUNCTION), 数组 (ARRAY), 集合类型 (SET), 用户自定义类型 (USER DEFINED), 记录 (RECORD), 文件 (FILE), 指针 (POINTER) 的程序, 例如 PF05B表示第五章过程和函数中第五个例题的第二个程序。

目 录

序.....	(7)
前言.....	(8)
阅读须知.....	(10)
第一章 基础知识.....	(1)
§1. 计算机基础.....	(1)
§2. 程序设计基础.....	(2)
§3. PASCAL 语言简况.....	(3)
§4. PASCAL 语言的程序结构.....	(4)
§5. PASCAL 语言的基本符号.....	(7)
第二章 标准的数据类型.....	(12)
§1. PASCAL 语言的数据类型.....	(12)
§2. 整数类型.....	(13)
§3. 实数类型.....	(14)
§4. 字符类型和布尔类型.....	(16)
§5. 标准函数.....	(18)
第三章 简单的程序设计.....	(25)
§1. PASCAL 语言的基本运算.....	(25)
§2. 常量定义和变量说明.....	(28)
§3. PASCAL 的语句类型.....	(30)
§4. 赋值语句.....	(31)
§5. 输入及读语句.....	(32)
§6. 输出及写语句.....	(35)
§7. 字符的输入.....	(41)
§8. 简单程序举例.....	(45)
第四章 流程的控制.....	(53)
§1. 如果语句.....	(53)
§2. 复合语句.....	(57)
§3. 情况语句.....	(58)
§4. 标号说明和转移语句.....	(65)
§5. 当语句.....	(69)
§6. 直到语句.....	(71)
§7. 循环语句.....	(73)
§8. 多重循环语句.....	(79)
§9. 退出循环语句.....	(82)
§10. 应用举例.....	(87)

第五章 过程和函数	(109)
§1. 标准过程和标准函数.....	(109)
§2. 简单过程和简单函数.....	(110)
§3. 过程说明和过程语句.....	(112)
§4. 函数说明和函数调用.....	(116)
§5. 全程变量和局部变量.....	(119)
§6. 数值参数和变量参数.....	(125)
• §7. 嵌套和递归.....	(130)
• §8. 过程参数和函数参数.....	(147)
• §9. 外部过程和外部函数.....	(154)
§10. 应用举例.....	(161)
• §11. 堆栈技术的应用.....	(175)
第六章 用户自定义数据类型	(181)
§1. 枚举类型.....	(181)
§2. 子界类型.....	(192)
第七章 构造型数据类型	(201)
§1. 集合类型.....	(201)
§2. 数组类型.....	(211)
§3. 记录类型.....	(225)
§4. 应用举例.....	(242)
第八章 文件	(271)
§1. 文件概述.....	(271)
§2. 文件的说明.....	(272)
§3. 用于对文件进行操作的的标准过程和标准函数.....	(274)
§4. TEXT 文件.....	(279)
§5. 几个简单的例证程序.....	(286)
第九章 指针	(298)
§1. 从静态变量到动态变量.....	(298)
§2. 指针.....	(301)
§3. 两个标准过程 NEW(P)和 DISPOSE(P).....	(305)
§4. 指针应用的几个简单程序.....	(306)
§5. PASCAL 程序与 MACRO 一级的程序的衔接.....	(328)
第十章 PASCAL 程序的建立、运行和调试	(335)
§1. PASCAL 源程序的建立与修改.....	(337)
§2. PASCAL 程序运行的步骤和方法.....	(344)
§3. 查找并改正 PASCAL 源程序中的错误.....	(366)
第十一章 综合应用程序举例	(392)
附 录	(453)
附录一、PASCAL 语法图.....	(453)
附录二、保留关键字.....	(457)

附录三、标准标识符.....	(458)
附录四、ASCII 字符集.....	(458)
附录五、OMSI PASCAL-1 语言规范.....	(459)
附录六、OMSI PASCAL-1 程序编译及运行的错误信息表.....	(464)
附录七、中英用语对照表.....	(468)
参考文献	(478)

第一章 基础知识

这一章是为全书提供一点基础知识。前二节仅仅扼要地提示一下计算机硬件和软件的基本梗概。确切地说，不是具体讲解有关的基础知识，因为学习 PASCAL 语言程序设计及应用的绝大多数读者，在这一方面都已有了一定基础。本章的后三节是关于 PASCAL 语言的基础知识部分，分别叙述了 PASCAL 语言的发展情况、程序的一般结构以及使用的基本符号。这些介绍为读者提供 PASCAL 语言的粗浅的概貌，为阅读后续章节做一点准备。

§1. 计算机基础

自从1946年第一台电子计算机问世以来，迄今为止才有37年历史，但是电子计算机工业已发展成为强大的独立工业部门，而且还在以惊人的速度不断发展。计算机的迅速发展和广泛应用已经深刻地影响到工农业生产、科学技术和社会生活各个领域。计算机在国民经济各部门中的应用可以归纳为三个主要方面。第一，它能高速地高精度地完成各种数学计算，被广泛地应用在科学研究和工程设计方面。第二，它能存储大量的数据信息，并能进行快速处理，这在信息处理、企业管理和情报检索等方面有广泛应用。第三，在自动控制方面，它代替人们对某些工农业生产过程进行监测和控制，提高产品的质量，减轻劳动强度，提高劳动生产率，或免使人们进入有毒或污染环境，保护人类健康。

一个完整的计算机系统由硬件和软件两部分组成。组成计算机的物质设备，我们称它为计算机的硬件。为使用计算机和发挥计算机效率功能的各种经常起作用的程序，我们称它们为计算机软件。也有人把计算机系统中除了硬件以外的东西，统统都称为软件。

计算机系统硬件主要由两部分组成，即处理机和外围设备。也有人概括为处理装置、输入装置和输出装置三部分。处理机包括存贮器、运算器和控制器。其中存贮器(简称为“内存”，也叫“主存”)在一定程度上决定着计算机总的运算速度和精度，决定着计算机能随机访问的存贮容量的大小。存贮器主要有磁芯存贮器和半导体存贮器，国外已经普遍使用半导体存贮器，磁芯存贮器已很少使用。内存贮器由于容量有限，不可能记录下全部输入信息，要借助辅助存贮器即外存贮器(通常是磁盘，磁带或磁鼓等)来记录大量暂时不用的信息。外存容量可以是内存容量的几千几万倍。计算机外围设备包括输入装置、输出装置和外存设备。在输入装置中，比较原始的是纸带输入机，国内常叫光电输入机。另一种是卡片输入机。这两种输入装置都要通过中间媒介。也就是在把信息送入计算机之前，要先作一段准备工作，把信息记在纸带或卡片上。而键盘输入装置可以在人手按键盘时，使计算机立即把信息收到机器内。这种输入装置对于人-机进行对话方式工作时特别合适。PDP-11 系列计算机系统，都有这种键盘输入设备，它还和显示终端相联系。当然输入方式还有语音输入，图象输入等各种各样的输入装置。输出装置也是多种多样的，大家比较熟悉的有凿孔机，终端显示器，宽行打印机等等。

在今天，光有计算机硬件本身，而不配备较完善的软件，要发挥机器的系统功能是不可能的。软件的作用在于：充分地发挥机器硬件的功能；使用户更为方便和有效地使用计算

机，对计算机控制的对象进行更理想的控制，以达到预期的目的；能诊断机器各部件的故障，并显示或打印出来，供维护人员及时排除；等等。软件分为系统软件、应用软件两大类。系统软件主要是指为实现计算机系统各种功能的有关软件，包括各种计算机语言，编译程序，操作系统，机器维护程序，服务性程序和装配程序等。专用软件，即应用软件，是指为实现控制或数据处理，完成某种计算的专门软件，包括专用语言和应用程序等。对于一般用户，了解和使用软件更为重要。

§2. 程序设计基础

程序设计是伴随着电子计算机的出现而产生的一门学科，简单地说，就是人们把需要计算机做的工作写成一种计算机能直接或间接接受的程序。程序设计的方式和水平不断改善，日益提高。程序所用的语言从简单到复杂，从低级到高级。象计算机硬件的发展一样，程序设计即计算机软件也有它的几代发展史：

第一代，机器语言和汇编语言。在计算机问世的初期，用二进制码表示计算机指令系统，用二进制代码编写程序——这就是“机器语言”。由于机器语言使用很不方便，编写这种程序极其繁琐，大大阻碍了计算机的广泛应用。为此人们用一些简单而又形象的符号来代替每一条具体的指令，而这些指令又对应于具体机器的二进制指令码，这就形成了“符号语言”。在此基础上，把一些子程序，存储器地址等也用符号来表示，这就是现在人们称呼的“汇编语言”。从汇编语言到机器语言，中间要有一个翻译过程，这便是翻译程序——叫“汇编程序”，简称“汇编”。机器语言和汇编语言，它们是与具体所用的计算机（确切地说是与计算机指令系统）相关的，是为特定的机器服务的，所以称为面向机器的语言。

第二代，高级语言阶段。人们在汇编语言的基础上，设想出能否避开具体的机器，用一些符号来描述自己的解题意图，尽量接近于数学公式的原始描述，而能够通过各类机器对应的翻译程序即可以在各类机器上运行。这便出现了各种高级语言。目前国内外比较通用的计算机语言有几十种。常见的应用最普遍的有 BASIC, FORTRAN, ALGOL, COBOL, PL/1 以及 PASCAL 等。在多数系统中，BASIC 语言的源程序是通过 BASIC 解释程序边解释边执行的。而其它几种语言要经过各自的编译程序编译之后，才能正常运行。因为计算机本身是不懂得这些语言的，要通过一段翻译工作才能把用一种语言写成的源程序与机器语言对应起来，这段翻译工作就是这种语言的编译程序。请注意，如果接了另一型号机器，则编译程序就又不一样了。

由于计算机系统迅速发展，运算速度不断提高，外部设备越来越丰富，主机功能日益完善，大容量的外存储器不断出现，……产生了“多道程序运行”和“分时操作”等新的概念，为有效地管理软、硬件资源，出现了高级的管理程序——操作系统。

第三代，为解决各种应用问题而设计了专用语言。随着计算机系统功能的发展，应用越来越广泛，各种专用语言会越来越多，人们的意图将更容易被计算机所接收，从而达到人们预想的目的。

对于学习 PASCAL 语言程序设计的大多数读者来说，程序设计基础主要是指掌握 PASCAL 语言的基本概念，熟练地编制有关的应用程序。至于编译程序和操作系统，已超过基本要求的范围，本书只是介绍了一些思路和例题，要真正掌握它们还得参考其它书籍，还得花很大气力。

一般程序设计，大致要经历以下几个步骤：

1. 根据任务提出问题；
2. 确定方法——构造数学模型，选择计算方法；
3. 编制程序——画出框图，编写程序，上机反复调试；
4. 在实践中检查程序的结果，如果正确无误，则设计基本完成。否则，要反复修改，直到满足要求为止。

程序员的职责，就是编制高质量的程序。所谓高质量的程序，其基本要求是可靠正确和简单清晰。同时在满足结构程序设计模块化要求的前提下，尽量做到程序短，占据内存单元少，程序运行时间省；还应该做到程序便于修改、调试，通用性好，可移植等。

§3. PASCAL 语言简况

PASCAL 程序设计语言，是瑞士苏黎世联邦工业大学沃斯教授于1968年研究出来的，最早发表在1971年瑞士的《ETH》杂志上。这种语言比 FORTRAN 语言，ALGOL 语言和 BASIC 语言等问世得都要晚。

PASCAL 语言是按照结构程序设计的原则设计的一种描写算法的语言。它是从 ALGOL 60 语言发展过来的，但是作了改进，效能更强。它适用于教学、管理、编写各种系统软件 and 进行科学计算。

PASCAL 语言有丰富的数据类型，简明的通用语句，清晰的程序结构，而且书写格式自由、编译紧凑，风度优美。因此博得了人们的好评，使用日益广泛。据了解，世界各国的许多高等院校都用它进行程序设计的教学，效果良好。清华大学几年来用 PASCAL 语言进行教学和科研的实践表明，PASCAL 语言受到特别欢迎。

关于 PASCAL 语言的特征，我们不妨把沃斯教授文章中的一段话摘录在下面，供大家参考。

“我们把 PASCAL 的特征列举如下：

1. 变量说明是强制性的。
2. 某些基字（例如，BEGIN，END，REPEAT 等）是‘保留’的，不能作标识符。
3. 分号（；）看作是语句分隔符，而不是语句终止符（如同 PL/I 语言一样）。
4. 标准数据类型是整数、实数、字符和布尔类型。基本的数据构造工具有数组、记录（对应于 COBOL 和 PL/I 语言的‘结构’）、集合和文件。这些结构可以组合和嵌套以形成集合的数组、记录的文件，等等。数据还可以动态地分配并由指针访问。
5. 集合 SET 数据结构提供了类似于 PL/I 语言的‘位串’的能力。
6. 数组可以具有任意维数与任意的界；数组的界是常量，即没有动态数组。
7. 象 FORTRAN，ALGOL 和 PL/I 语言一样，存在转移语句。标号是无符号整数，必须加以说明。
8. 复合语句如同 ALGOL 语言，相应于 PL/I 语言里的 DO 群。
9. ALGOL 语言的开关和 FORTRAN 语言的计算转语句的功能在 PASCAL 中由分情形语句表示。
10. 循环语句相应于 FORTRAN 的 DO 循环，它的步长只能是 1(TO) 或 -1(DOWN-TO)，并仅当控制变量的值落在界内时执行。因此，被控制的语句可能完全没有执行。

11. 没有条件表达式和多重赋值。

12. 过程和函数可以递归调用。

13. 对变量不存在 (象ALGOL语言里那样) 'OWN'属性。

14. 参数有值调用和引用调用, 没有名字调用。

15. 就不存在匿名的分程序来说, 这里的'分程序结构'不同于ALGOL和PL/I语言的分程序结构, 即每个分程序都给一个名字, 并做成一个过程。

16. 常量、变量等所有对象必须在它们被引用之前加以说明。但允许以下两个例外:

(1) 指针类型定义里的类型标识符;

(2) 当存在向前引用时的过程与函数调用。

许多人初次接触PASCAL语言, 往往感叹它缺少某些'中意的特色', 例如方幂运算符、串的并列、动态数组、布尔值的算术运算、自动的类型转换和省缺说明等, 这都不是疏忽遗漏, 而是有意删除的。在某些场合下, 有它们反而会招致程序设计解法的效率不高; 在另外一些场合下, 又会感到它们不利于达到程序设计清晰可靠和'风度优美'的目标。最后, 还得考虑对繁多的有效的程序设计功能进行严格挑选, 以保证编译程序比较紧凑和高效。”

目前, 国际上PASCAL语言有许多种版本, 例如有:

一、标准PASCAL语言 1971年, 沃斯教授提出并定义了PASCAL语言。1975年, 他和金埃森 (J.Jensen) 对PASCAL语言进行了修改, 修改报告作为“标准PASCAL语言”。主要文献是《PASCAL MANUAL AND REPORT》。

二、PASCAL 8000语言 它是标准PASCAL的扩展版本。1976年, 由日本东京大学的HIKITA和ISHIHATA用PASCAL语言编写编译程序。它能在OS7操作系统控制下应用于HITAC8800/8700计算机系统 (与IBM/370指令系统相同)。后来, 这个版本经过澳大利亚原子能委员会 (AAEC) 的COX和TOBIAS的修改和扩展, 能在OS操作系统控制下应用于IBM360/370计算机。PASCAL 8000语言版本的主要参考书是《PASCAL 8000 REFERENCE MANUAL》。

三、OMSI PASCAL语言 它是由美国俄勒冈州小型计算机软件公司 (OMSI) 确定, 在RT-11操作系统控制下应用于PDP-11/03等计算机系统上。主要参考资料是:

1. 《OMSI PASCAL-1 LANGUAGE SPECIFICATION》
2. 《OMSI PASCAL-1 USERS GUIDE FOR RT-11》
3. 《OMSI PASCAL-1 DEBUGGER》

本书是以标准PASCAL语言的有关规定为指南编写的。但全部程序都是用OMSI PASCAL-1语言编写的, 并在PDP-11/03计算机系统上进行调试和运行。这两个语言版本有一些具体区别。一方面, OSI PASCAL-1语言缺少标准PASCAL语言的个别功能, 如过程PACK和UNPACK。另一方面, 它对标准PASCAL语言进行了许多扩展。这些扩展增强了该语言的性能, 允许用户更有效、更容易和更清楚地描述算法。

§4、PASCAL语言的程序结构

一个程序语言的基本功能是对数据进行描述和操作。一个程序, 从本质上讲, 它是描述对给定数据的处理过程。

PASCAL语言是如何描述数据又是怎样对数据进行操作的呢? 用PASCAL语言编写

的程序，其一般格式如何？这是用户首先关心的问题。

我们从模仿开始，先看一个简单的例题。题意是输入两个整数，计算并输出该两数之和。程序如下：

```
PROGRAM SIMPLE (INPUT, OUTPUT) ;  
  {THIS PROGRAM IS USED FOR S=X+Y}  
VAR  
  X, Y, SUM: INTEGER;  
BEGIN  
  READ (X, Y) ;  
  SUM:=X+Y;  
  WRITELN ('SUM=', SUM)  
END.
```

在这个简单程序中，实际上分为三部分。其中：

第一部分是程序的首部，包括第一节二行。

它指出这是 PASCAL 语言的程序，其名称为 SIMPLE（简单的源程序）；程序的参数有两个：INPUT, OUTPUT。程序的功能是进行 X 加 Y 的运算。

第二部分是程序的说明部分，包括第三第四行。

它指出程序中使用了三个变量：X, Y, SUM。它们都是整数类型的变量。

第五行至第九行是程序的执行部分。

它指出这个程序做三件事：

- (1) 通过读语句 READ (X, Y)，输入整数 X 和 Y；
- (2) 通过语句 SUM:=X+Y，计算两个整数 X 和 Y 之和；
- (3) 通过写语句 WRITELN ('SUM=', SUM)，输出计算结果。

事实上，用 PASCAL 语言编写的源程序一般都有这三部分，只不过其内容有多有少。

PASCAL 语言的程序结构，见附录 A，其示意图如图 1.4-1。

从图 1.4-1 可以看出，PASCAL 语言的程序结构是模块结构，层次分明，清晰整齐。一般可以分为三大部分。

一、程序首部 程序首部是程序的开头部分。它必须提供程序的主要特征，一般由四个小部分组成。

1. 程序的标志。

PASCAL 语言规定，程序一律以 PROGRAM 开头，作为程序的标志。

2. 程序的名称。

程序的名称由用户自己定义。名称可以形象化一些，也可以随便取。例如程序是关于拷贝文件的，由李明同志编写的，可以取名称为 COPYFILE，或 LIMING01。本书中的程序名称，根据章节分类名加序号而定。比如第三章简单的程序设计，其举例程序的名 称 取：SMP01, SMP04……。第八章、文件，例证程序取名为 FIL01, FIL02A 等。

3. 程序的参数。

程序的参数用来表示该程序与外界的联系。这些参数一般是文件变量名。程序通过这些参数调用外部文件。最常用的程序参数为 INPUT, OUTPUT。它表示该程序有输入及输出的操作。

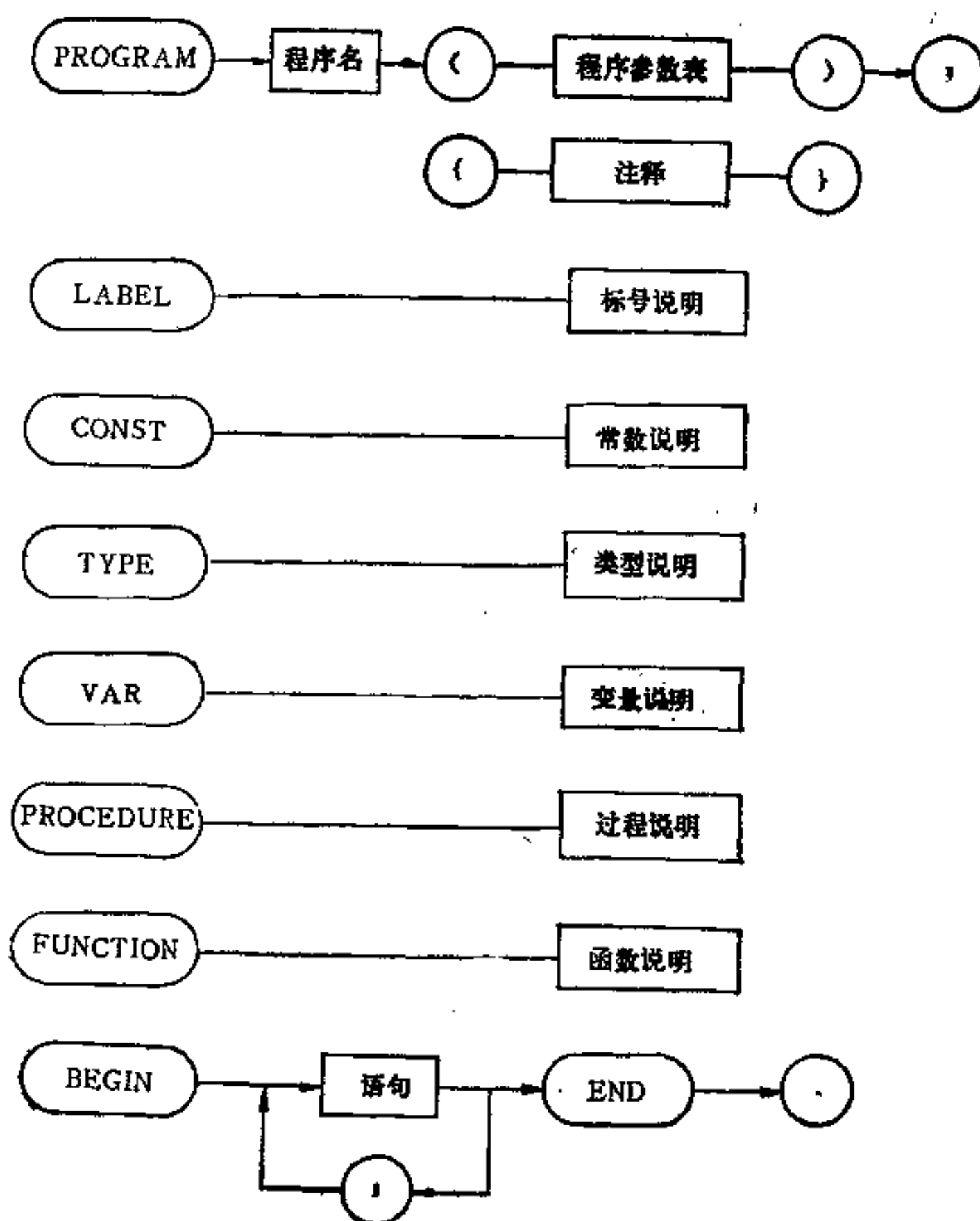


图 1.4-1 程序的结构

4. 程序的注释部分。

注释部分一般用来说明该程序的名称，类型，主要功能，编写日期等等。没有规定，也没有限制，由用户自己决定要注释的内容。它仅供用户阅读，机器并不执行。注释还可以出现在程序的任何位置，只要不忘记正确使用注释符即可。

二、程序说明部分 PASCAL 语言本身定义了一些标准的常量，标准的数据类型，标准的过程和函数。这些标准的内容可以直接使用。

PASCAL 语言允许用户自己定义标号，常量，类型，变量，过程和函数等。这些都必须首先在程序的说明部分加以说明，然后才能在程序的执行部分引用。在上一节提到 PASCAL 语言的特征列举的第16条“常量，变量等所有对象必须在它们被引用之前加以说明”，就是说的这一点。

程序的说明部分，在说明时应遵循如下次序：

1. 〈标号说明部分〉；
2. 〈常量定义部分〉；

3. 〈类型定义部分〉；
4. 〈变量说明部分〉；
5. 〈过程与函数说明部分〉。

三、程序的执行部分 在图 1.4-1 程序的结构中，BEGIN 和 END 之间的部分为程序的执行部分。它由一系列语句组成。每一条语句执行一定的动作，完成一定的任务。两个语句之间用分号（；）隔开。整个程序用实心句号（.）作为结尾。

PASCAL 语言比某些语言的突出之点是格式自由，非常灵活。允许一行写几个语句，也允许一个语句写成几行。这一点比 BASIC 和 FORTRAN 语言要方便些。

OMSI PASCAL 语言在程序结构上更灵活。它规定：

1. 程序的首部可以省去；
2. 程序的参数可以不写；
3. 注释部分允许有三种注释符，即有三种格式表示：
 - (1) { }；
 - (2) (* *)；
 - (3) / * * /。

因此，用 OMIS PASCAL 语言写的最简单的程序甚至可以只有执行部分。例如：

```
BEGIN
    WRITELN (SQR(15))
END.
```

这个程序运行后会输出整数 15 的平方为 225。

当然，作为一个完整的程序，应该写上程序的首部。这样做，有利于识别和区分不同的程序。而且在编译的过程中列出清单时，每一页都会显示程序名称。

注释部分似乎没有多大用处。但这是误解。在复杂的程序中加上注释部分是非常有益的。养成注释的习惯，对今后阅读、修改程序十分有利。因为有了注释，可以增加程序的可读性。尤其在程序的难点或关键之处，注释可以给人启示，使人一目了然。这样做，便于程序进行交流。注释不仅可以写在程序首部之后，而且可以写在程序的任何位置。

注释部分的结构如下：

```
{ 〈任意不包含“}”的字符串〉 }；
或( * 〈任意不包含“*”的字符串〉 * )；
或/ * 〈任意不包含“*/”的字符串〉 * /。
```

它可以扞在任何两个标识符、数或特殊符号之间。但注意单个符号“{”（或“(*”，或“/*”）和“}”（或“*)”，或“*/”）不能出现在语言的其它部分。

注释部分删除后，程序的意义不变。

这一节，我们仅仅概括地介绍了 PASCAL 语言程序的总体结构，后续章节将详细地分析这些问题。

§5、PASCAL 语言的基本符号

任何一个计算机系统所能使用的字符都是固定的、有限的。它要受设备的限制。

一、基本符号的分类 在 PASCAL 语言中，基本的符号有三类：

第一类，大写英文字母，共有26个字符：

A B C D E F G H I J K L M N O P Q R S
T U V W X Y Z

第二类，数字符号，共有10个字符：

0 1 2 3 4 5 6 7 8 9

第三类，特殊符号，其中又分运算符和分界符。

1. 运算符，由专用字符组成，共24个：

+ - * / := ↑ = <>
<= >= < > () []
{ } : ' " , ; .

2. 分界符，又称保留关键字，它们是具有固定意义的单独符号，共有35个：

AND	ARRAY	BEGIN 开始	CASE
CONST 常量说明	DIV 整除运算符	DO	DOWNTO
ELSE 否则	END 结束	FILE	FOR
FUNCTION 函数说明	GOTO	IF 如果	IN
LABEL 标签说明	MOD	NIL	NOT
OF	OR	PACKED	PROCEDURE 过程说明
PROGRAM 程序	RECORD	REPEAT	SET
THEN 那么	TO	TYPE	UNTIL
VAR 变量说明	WHILE 循环体	WITH	

二、标识符 标识符，就是用来表示程序、过程、函数、类型、常量和变量等名称的符号。

例如，在上一节例题程序中，SIMPLE 表示程序的名称；X，Y和SUM表示变量的名称。它们都符合标识符的规定。

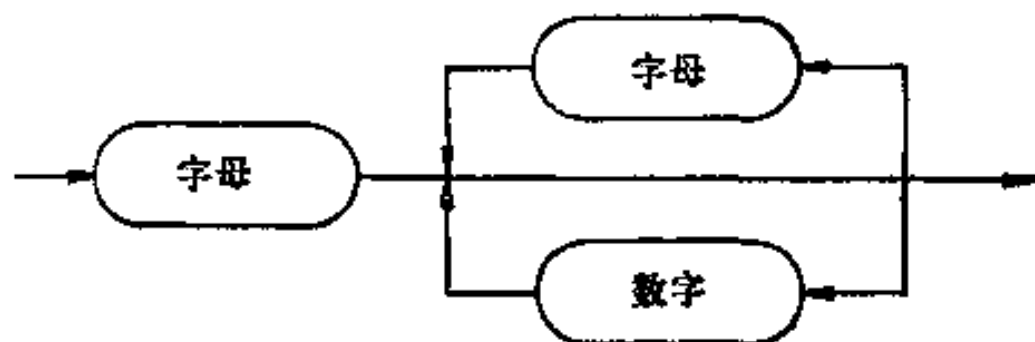


图 1.5-1 标识符的格式

标识符规定，标识符必须以字母开头，后面可以跟数字或字母组成的任意序列，也可以不跟任何字符，单独一个字母构成标识符。其格式如图1.5-1。

例如：X，SUM，SMP05，CTL16等是合法的标识符。而1A，A+B，P*Q，SMP 05，LEVEL.4等是非法的标识符。

一般地说，一个标识符可以很长，但是实现时可以限定有效字符的个数。标准PASCAL语言在实现时把一个标识符的前八个字符看作是有效的。因此，表示不同的标识符应在前八个字符中有所区别。

例如：THISISAPASCALPROGRAM，THEFIRSTBOOK和THATSECONDDISK都是合法的标识符。如果还有两个标识符：THEFIRSTDISK，THATSECONDDPRINTER，虽然本身也是合法的，但是，标准PASCAL语言在实现时就无法区别了，因为第二个和第四个标识符，第三个和第五个标识符，它们的前八个字符是相同的。

在PASCAL语言中，有一些特殊的标识符，通常称它们为予定义的标识符，即PAS-

CAL语言本身选用了的标识符，其中一类是保留关键字 (RESERVED WORDS)，另一类称为标准标识符 (STANDARD IDENTIFIERS)。

保留关键字，它们有特殊规定的意义，绝对不允许用户将它们作为别用，它们是用户定义标识符的“禁区”。一旦用户擅自拿作他用，程序运行时一定出错。

保留关键字一共有35个，见附录二所示。

标准标识符，它们也具有特殊规定的意义，一般情况下同样不允许用户作为他用，它们是用户定义标识符的“危险区”。一旦用户把它们定义为另外意义的标识符，则使它们失去原来的意义。例如，在某个程序中作如下定义：

```
CONST
```

```
SIN=15;
```

即定义常量标识符SIN为整数15。如果在该程序中不涉及正弦函数的问题，那末，程序一般仍可以正常运行。一旦在程序中涉及到正弦函数的问题，则该程序一定不能正常运行。在进行求正弦函数值的地方，指出错误：缺少操作符。这是因为原来求正弦函数值的操作符SIN，已经变成是另外的意义，SIN已是一个常量15的标识符了。

标准标识符，一共有40个，其中常量的标准标识符有FALSE, TRUE, MAXINT三个；标记类型的标准标识符有INTEGER, REAL, CHAR, BOOLEAN, TEXT五个；文件的标准标识符有INPUT, OUTPUT两个；函数的标准标识符十七个，过程的标准标识符十三个。详见附录三。

三、分隔符

分隔符是算法语言中的又一个重要概念。在PASCAL语言里，把空白符、行结束符和注释看作分隔符。

PASCAL语言规定，在任意两个（或称为一对）相邻的标识符、数或单词符号之间，必须至少有一个分隔符。但是在一个标识符、数或特殊符号的内部不允许出现分隔符。例如：

```
PROGRAM SIMPLE01;
```

```
VAR
```

```
X, Y, SUM: INTEGER;
```

这是合法的。但是将上述程序首部和变量说明写成如下格式：

```
PRO GRAM SIMPLE 01;
```

```
VAR,
```

```
X, Y, SUM: INTE GER;
```

这样就是非法的了。原因是将PROGRAM、SIMPLE01和INTEGER都拆开了，而在VAR之后又乱加了逗号(,)。

四、OMSI PASCAL语言的一些特殊规定

1. 在标识符问题上，OMSI PASCAL语言允许标识符为任意长度，并且全部字符都是有意义的。它允许用小写体字母，并且自动转换为对应的大写体字母。因此THEFIRSTBOOK和THEFIRSTDESK是可以区别的两个标识符。如果在标记标识符打了小写字母abc，它会自动转换为ABC。

由于目标程序文件结构的限制，用于EXTERNAL和FORTRAN过程的标识符的前六个字符必须是单值的，可区别的。否则编译时会出错。

2. 在文件名称及其类型的规定上，在使用PDP-11系列机时，应遵循文件名的有关

LABEL, ✓	ST53D x	12AE, x	ENGLISH, ✓
PDP-11; x	SIN, ✓	UNTIL, ✓	WRITE, ✓
BEGINWORK. ✓			

第二章 标准的数据类型

一个计算机程序包括两个本质的部分，一个用于描述数据，另一个用于描述对数据所进行的操作。描述数据由类型定义和变量说明来实现。在 PASCAL 语言里，程序的说明部分解决这个问题。描述对数据所进行的操作，由一系列执行语句来实现。在 PASCAL 语言里，程序的执行部分解决这个问题。

本章一开始讨论数据类型。首先介绍四种标准的数据类型，然后介绍与之有关的标准函数。

§1. PASCAL 语言的数据类型

PASCAL 语言的优点之一是有丰富的数据类型。按照它们的特点，可以分成三大类。

一、简单的数据类型 这一数据类型构造简单，又称为“非构造型数据类型”。它包括常用的四种标准数据类型：

整数类型；

实数类型；

字符类型；

布尔类型。

它还包括两种用户自定义的数据类型：

枚举类型；

子界类型。

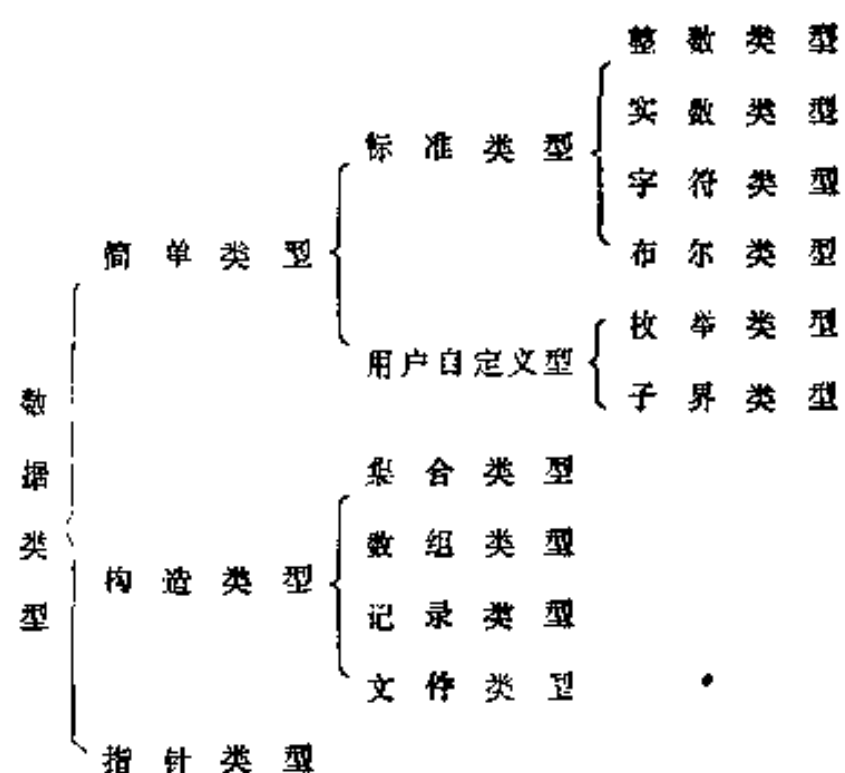


图 2.1—1 PASCAL 数据类型

二、构造型数据类型 构造型数据类型，又称为复杂数据类型，其构造复杂，一般由其它数据类型按照一定的规则构造而成。它包括常用的四种数据类型：

集合类型；
数组类型；
记录类型；
文件类型。

其中由10个字符组成的数组被定义为标准的数据类型 ALFA；由字符组成的文件被定义为标准的数据类型 TEXT。

三、指针类型 这是一种使用灵活的简单数据类型，它与第一、第二类的数据类型不同，主要用于解决动态数据的建立、删除和使用等问题。

本章仅仅讨论四种标准的数据类型，其它类型的数据将在第六章到第九章讨论。

关于 PASCAL 语言的数据类型的分类如图2.1-1所示。

§2. 整数类型

整数类型数据包括正整数，负整数和整数零。其格式如图2.2-1。

在 PASCAL 语言中，数一般以十进制表示，数字 0 到 9 都可以使用。在特殊情况下，可以通过说明，用八进制或十六进制表示。

其中正整数或整数零可以不写符号位。因此，例如+123与123相比，具有同等意义。

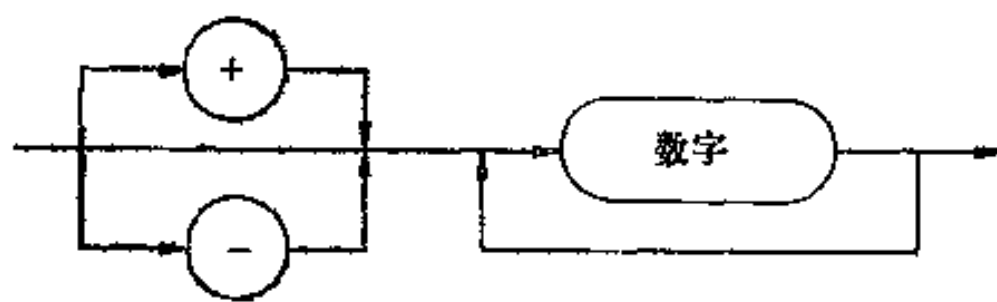


图2.2-1 整数格式

一般的整数只能由正负号和数字序列组成，数字序列中间不允许出现任何其它符号。

例如：-13579； 0； 1375等是合法的整数。而-157A4； 210.0； 12,456等是非法的。因为-157A3不能代表任何整数；210.0则不能代表整数，实际上是一个实数；12,456在数学上是允许的，但在PASCAL语言中是非法的。

在 PASCAL 语言中，有一个特殊的整数，用 MAXINT 表示。它代表某个计算机系统允许的最大整数值。整数的范围一般是由计算机的字长决定的。因此，在不同的计算机系统里，MAXINT 的数值可以是不相同的。

在 PDP-11/03计算机系统里，字长为16位。定义 $\text{MAXINT} = 2^{15} - 1$ ，MAXINT 的数位为32767。

在 CDC 6000计算机系统里，字长为60位。为了给整数运算留点余地，定义 $\text{MAXINT} = 2^{48} - 1$ ，MAXINT 的数值为281474976710655。

整数不仅有最大值，也规定了最小值。它同样受计算机系统的限制。

在 PDP-11/03计算机系统中，整数的范围是 $(-32768 \dots 32767)$ 。

在 CDC 6000计算机系统里，整数的范围为 $(-281474976710656 \dots 281474976710655)$ 。

在 OMSI PASCAL 语言里，如果输入的数据不符合整数的规定，则在编译程序时，指出“整数有错” (BAD INTEGER)。如果运算的结果，例如加法、乘法和乘方运算的结果，超出整数范围，则在运行时指出“整数错误” (INTEGER ERROR)。

在 PASCAL 语言中, 只能用类型标识符 INTEGER 来说明整数类型变量。
例如在前面举过的例题程序中, 变量 X, Y 和 SUM 都属于整数类型, 则可以说明如下:
VAR

X, Y, SUM: INTEGER;

在使用整数过程中, 还会有其它的问题, 在以后的章节中再作进一步讨论。

§3. 实数类型

实数类型的数据包括正实数, 负实数和实数零。

一. 实数的两种表示法 在 PASCAL 语言中, 一般用两种方法表示实数: 十进位表示法和科学表示法。

1. 十进位表示法

十进位表示法就是小数形式的表示方法, 它是人们日常生活中习惯使用的方法。其格式如图 2.3-1:

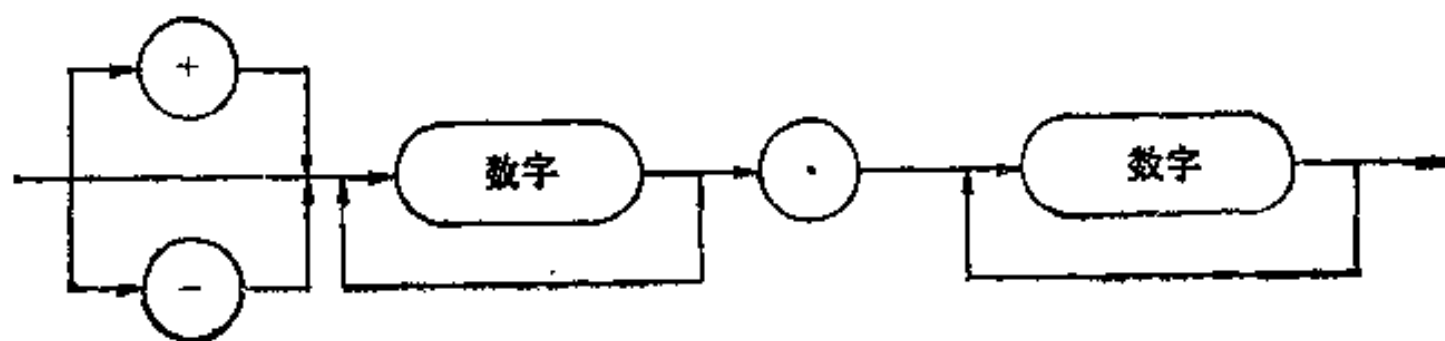


图 2.3-1 实数的十进位表示法格式

其中:

小数点左侧为实数的整数部分;

小数点右侧为实数的小数部分;

正实数和实数零可以不写符号位。

例如, -3.5 ; 0.0 ; -0.0 ;

-123456.789 ; $+98765.4321$ 等等,

都是合法的实数型数据。

在使用十进位表示法时必须注意两点:

第一, 一个整数可以当作实数使用, 但是反过来不行。例如:

整数 1000 可以作为实数 1000.0 使用, 这时并不要求把 1000 写成 1000.0。

第二, 一个实数中只要有小数点, 则小数点两侧都必须有数字, 缺一不可。这跟人们的习惯及其它某些算法语言 (如 FORTRAN 语言) 不同。

例如, 在 PASCAL 语言里, 下面两个数: -4567 和 $+123.$ 的表示是非法的。

2. 科学表示法

科学表示法就是指数形式的表示方法, 它与计算机的浮点表示是一致的。其格式如图 2.3-2。

其中:

字母 E 表示以 10 为底的指数;

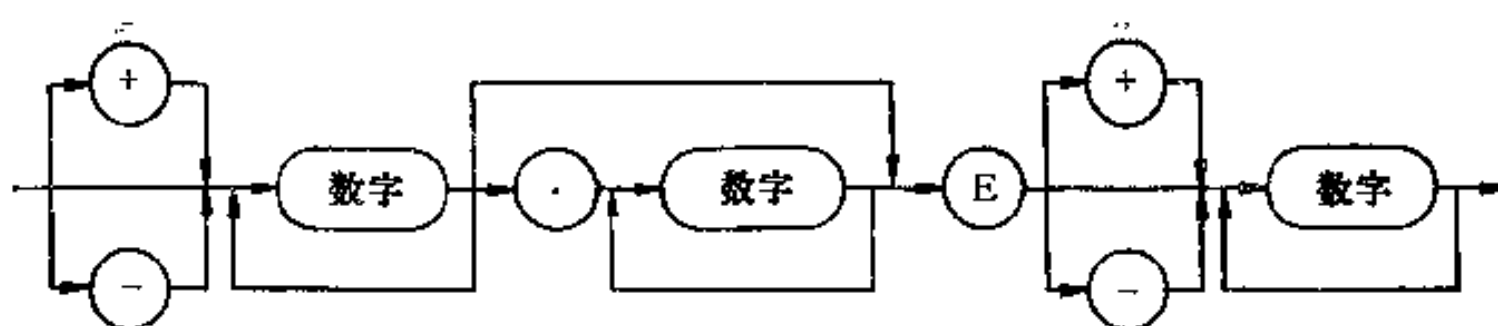


图2.3-2 实数的科学表示法格式

E的左侧表示实数的尾数部分，包括数符及尾数；

E的右侧表示的是实数的指数部分，包括阶符及阶码。

正实数及实数零可以不写数符，当一个实数的阶码为正整数或整数零时，可以不写阶符；

实数的尾数必须有，但可以没有小数部分；

实数的阶码必须有，而且必须是整数。

例如，下列实数，

$+5.0E+2$;	$+5.0E2$;
$5.0E+2$;	$5.0E2$;
$5E2$;	500.0 ;

它们都是合法的，而且是相等的。

请注意比较下述两对实数：

(1) $-6.08E-15$ 和 $6.08E+0.5$

(2) $-1.0E+2$ 和 $+E-2$

$-6.08E-15$ 和 $-1.0E+2$ 是合法的。

$6.08E+0.5$ 和 $+E-2$ 是非法的。前者阶码不是整数，后者缺少尾数部分。

实数的两种表示法各有优缺点。当一个实数的位数有限时，用十进位表示法比较清楚，人们一目了然。当实数位数太长，比如十位以上，十五位以上，那末用十进位表示就不方便了，而用科学表示法比较好。

二、实数的范围及运算精度 无论用十进位表示法还是科学表示法来表示实数，它们在计算机里总是用浮点方法来实现的。和整数类似，实数也有个范围问题。

在数轴上，一个实数可以表示如图2.3-3

其中：

N_1, N_2, N_3, N_4 都是正整数；

C区间和D区间实数绝对值太小，接近于零，产生“下溢”；

A区间和B区间实数绝对值太大，超过机器允许范围，产生“上溢”；

E区间和F区间，实数的绝对值适中，作为计算机表示的范围。对于不同的计算机系统，表示实数的方法不同，实数的范围也不相同。

例如 CDC 6000计算机系统里，字长60位，实数用一个字长表示。其中指数部分为12位，尾数部分为48位，相当于14位十进制数。定义 $N_1 = N_4 = 322$, $N_2 = N_3 = 294$ 。

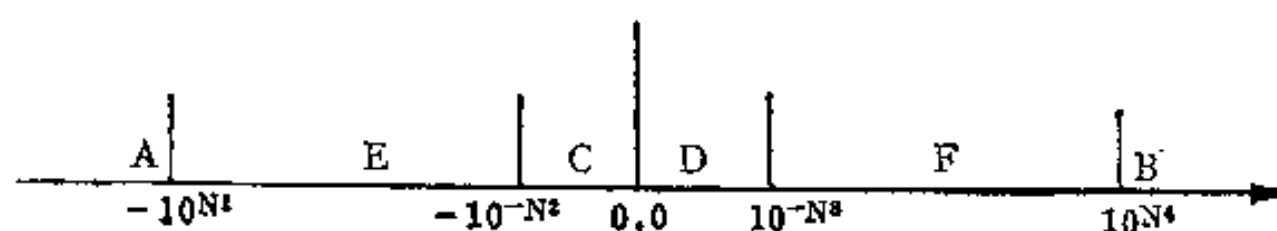


图2.3-3 实数在数轴上的表示

在 PDP—11/03 计算机系统里, 字长 16 位, 实数用二个字长表示。其中指数部分为 9 位, 尾数部分为 23 位, 相当于 7 位十进制数。这里, 定义 $N_1 = N_2 = N_3 = N_4 = 38$ 。

在 PDP—11/03 计算机系统中, 实数的范围为:

$(-1.0E+38 \dots -1.0E-38)$ 和 $(1.0E-38 \dots 1.0E+38)$ 。

CDC 6000 计算机系统实数范围则为:

$(-1.0E+322 \dots -1.0E-294)$ 和 $(1.0E-294 \dots 1.0E+322)$ 。

由此可见, 一个计算机系统的实数范围比整数范围要大得多。这给算术运算带来很大的便利。

例如, 在 PDP—11/03 计算机系统里, 不能把 345670 作为整数参加运算。但是, 如果把它变成实数 345670.0 或 $3.4567E+5$, 则可以进行实数运算。

实数运算与整数运算 (在允许范围内) 不同, 不仅运算速度减慢, 而且运算的精度有限制。比如, 在 PDP—11/03 计算机系统里, 最多可以提供七位十进制数, 其中前六位是有效数字, 第七位是后面四舍五入的结果。比如, 输入的实数是 1234.90765, 机器实际接受的是 1234.908, 最后一位 8 是进位的结果。如果输出这个实数, 得到的只能是 1234.908。

考虑到运算的速度和精度, 凡是可以用整数计算的问题尽量用整数计算。但是实际上由于整数的范围太小, 仅用整数计算的用途是很有限的。

在 PASCAL 语言中, 只能用类型标识符 “REAL” 说明实数型变量。

例如, 变量 Y, Z, AVERAGE 是属于实数类型, 则可以说明如下:

VAR

Y, Z, AVERAGE: REAL;

关于实数使用中的其它问题, 在以后章节中予以讨论。

§4. 字符类型和布尔类型

一、字符类型 字符类型数据包括全部可以打印的字符。在 PASCAL 语言中, 表示字符类型数据的格式如图 2.4-1

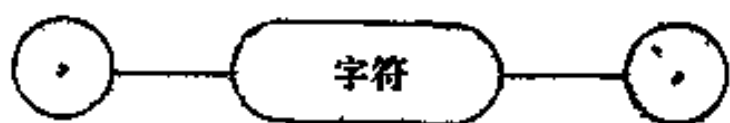


图 2.4-1 字符类型数据的格式

其中 “'” 称撇, 又叫单引号, 在字符格式中, 必须成对出现; 字符必须是可以打印的字符, 不能是控制字符。

例如, 下列字符是合法的字符:

'A', '8', '?', '□'

它们分别表示字符 A, 字符 8, 字符? 和字符空格。

“空格”是一个重要的字符, 它表示在该位置上留一个空格。为了表示清楚起见, 我们在本书中, 在表示字符“空格”的地方用 “□” 代替。实际上并不存在符号 “□”, 只是在相应的地方留一空格而已。

表示字符撇 (') 本身, PASCAL 语言规定要书写两撇 (')。例如, 字符'' 是合法的, 它表示字符撇 ('); 而 ' 是非法的字符, 它不能表示字符撇 ('), 也不能表示任何数据。

不同的计算机系统, 使用不同的字符集。例如, CDC6000 系统用 64 字符集。还有的用 256 字符集。PDP—11/03 计算机系统用的是 128 个字符的 ASCII 码。ASCII 码是国际上用得

最普通的字符集。它是美国标准信息交换码 (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE) 的英文字母缩写, 共有 128 个字符, 每一个字符对应一个字符码。字符码表示该字符排列的顺序 (次序号)。其中字符码 00 至 31 (共 32 个) 以及 127 所对应的字符是控制字符。控制字符 (如回车, 换行等等) 是不能打印出来的。其余 95 个字符可以打印。因此, ASCII 码可供打印的字符类型数据是 95 个。

附录四表示的是在 PDP-11/03 计算机系统上使用的 ASCII 码表。

尽管有好几种字符集, 但是最基本的字符有 36 个, 这是任何字符集都有的。它们是 26 个英文字母 (A..Z) 和 10 个阿拉伯数字 (0..9)。此外, 空格字符也是最常用的字符。

在计算机系统中, 字符是通过其对应的字符码 (转化为二进制码后) 来存贮的。字符码一般由一个字节 (8 位) 组成。

例如, 'A' 对应 65; '?' 对应 63; '_' 对应 95; ':' 对应 58; '8' 对应 56; 'X' 对应 88, 'x' 对应 120 等等。

计算机存贮 'A', 实际上存贮的是 01000001

计算机存贮 '?', 实际上存贮的是 00111111

计算机存贮 '_', 实际上存贮的是 00100000

计算机存贮 ':', 实际上存贮的是 00111010

计算机存贮 '8', 实际上存贮的是 00111000

计算机存贮 'X', 实际上存贮的是 01011000

计算机存贮 'x', 实际上存贮的是 01111000

一般情况下, 用一个字节存贮一个字符就可以了。特别是在那些按字节为基本单位寻址的计算机中, 这种办法是很合适的。有时, 在一些以字为基本单位寻址的计算机中, 为存贮方便, 也有用一个字来存放一个字符的, 这样做, 对内存单元来说是很大的浪费, 所以这种办法不常见。

在 PASCAL 语言中, 只能用标准标识符 CHAR 来说明字符类型变量。这里的 CHAR, 是英文 CHARACTER 的缩写。

例如, 如果 C 是字符类型变量, 则可以说明如下:

VAR

C:CHAR;

必须强调, 字符类型的数据只能是一个字符, 不能是一串字符。

例如, 'ABC', '789', 'P8=?' 都不是字符类型的数据, 它们不具有字符类型数据的性质。它们是字符串, 属于构造型数据。本书将在第七章数组一节中讨论。

二、布尔类型 布尔类型数据共有两个: TRUE 和 FALSE, 它们分别表示逻辑判断的结果是真 (TRUE) 或假 (FALSE)。

例如, 100 大于 10, 对吗? 回答是, 对的。那么可以表示为:

100 > 10 为 TRUE。

又如: 100 等于 10, 对吗? 回答不对, 错。那么可以表示为:

100 = 10 为 FALSE。

在 PASCAL 语言中, 布尔型数据与整数型数据和字符型数据一样, 也是有序的。而且规定:

FALSE < TRUE

在 PASCAL 语言里, 只能用标准标识符 BOOLEAN 来说明布尔类型的变量。

例如: 变量 SWITCH 表示电键的开关状态, 属于布尔类型, 则可以说明如下:

VAR SWITCH: BOOLEAN;

布尔类型数据个数最少, 比较简单, 但它的用途很广泛, 主要服务于程序设计中的流程控制和逻辑判断。

在 PDP-11/03 计算机系统中, 一个布尔型数据虽然只要用一个二进制位表示就够了, 但为了存取方便, 一般让它占用一个字节。

§5. 标准函数

在 PASCAL 语言中, 定义了多种标准函数。这些标准函数的名称已经确定, 作为 PASCAL 语言自身选定的标准标识符, 用户只要直接使用就可以了。这些标准函数的自变量只有一个, 用户可以自由确定。

标准函数的格式如图 2.5-1。

PASCAL 语言共定义了十七种标准函数。每一种标准函数都有自己的特性。这主要体现在两方面:

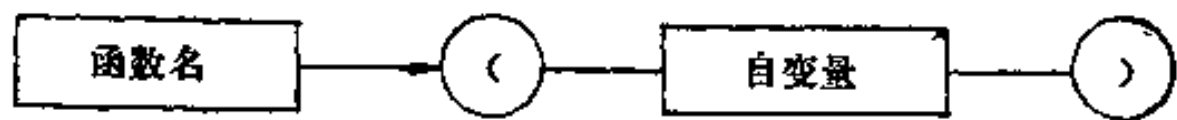


图 2.5-1 标准函数的格式

一方面, 它对自变量的数据类型有要求。比如, 字符类型的数据不能求其三角函数; 函数 CHR(x) 的自变量必须是整数, 而且范围很小; 函数 SQRT(x) 中的自变量不能是负数, 等等。

另一方面, 它对函数值的数据类型有规定, 以便使这些标准函数参加进一步的运算。

例如, 函数 ROUND(x) 和 TRUNC(x), 自变量数据类型是实数, 函数值一定是整数; 三角函数 ARCTAN(x), COS(x) 和 SIN(x), 自变量可以是整数或实数, 但函数值必定为实数, 等等。

PASCAL 语言定义的全部标准函数, 列表如下页。

按照函数的特点, 十七种标准函数可以分为四类: 算术运算函数, 逻辑判断函数, 转换函数和进退函数。

为了讨论方便, 我们定义自变量一律为 x, 若函数值为实数, 取四位小数。

下面分别讨论各类函数。

一、算术运算函数 这是常用的几个初等函数, 共有八种:

1. 绝对值函数: ABS(x)

ABS 取自 ABSOLUTE (绝对) 的缩写。

其中 x 为整数或实数, 函数值也为整数或实数。正数的绝对值就是其自己。负数的绝对值为其对应的正数。例如:

$$\text{ABS}(+3.1416) = 3.1416$$

$$\text{ABS}(-1000) = 1000$$

2. 平方值函数: SQR(x)

SQR 取自 SQUARE (平方) 的缩写。

其中 x 为整数或实数, 函数值也为整数或实数。无论 x 为正数还是负数, 它的平方值总是正数。x 为零值时, 平方值也为零。例如:

标 准 函 数 表

标准函数 函数值类型 \ 自变量类型	整 数 (INTEGER)	实 数 (REAL)	字 符 (CHAR)	布 尔 型 (BOOLEAN)	文 件 (FILE)
整 数 (INTEGER)	ABS SQR PRED SUCC	TRUNC ROUND	ORD	ORD	
实 数 (REAL)	SIN COS ARCTAN LN EXP SQRT	SIN COS ARCTAN LN EXP SQRT ABS SQR			
字 符 (CHAR)	CHR		PRED SUCC		
布 尔 型 (BOOLEAN)	ODD			PRED SUCC	EOF EOLN

$$\text{SQR} (+10.0) = 100.0000$$

$$\text{SQR} (-15) = 225$$

3. 平方根值函数: SQR T (x)

SQR T 取自 SQUARE—ROOT (平方根) 的缩写。

其中 x 必须为正数 (正整数或正实数), 平方根值一律为正实数。负数不能求平方根值。

例如:

$$\text{SQR T} (+100) = 10.0000$$

$$\text{SQR T} (1.21) = 1.1000$$

SQR T (-100) 在程序运行时出错。

4. 正弦函数: SIN (x)

其中 x 为整数或实数, 但函数值一律为实数。要注意的是, x 必须是弧度数, 不能直接用角度数。如果给的是角度, 必须进行转换。设角度为 y, 弧度数自变量 x, 那末:

$$x = \pi / 180 * y, \pi \text{ 值取其四位小数为 } 3.1416.$$

例如, 求 30° , 45° 的正弦函数值:

$$\text{SIN} (3.1416 / 180 * 30) = 0.5000$$

$$\text{SIN} (3.1416 / 180 * 45) = 0.7071$$

5. 余弦函数: COS (x)

其规定与正弦函数相同。

例如, 求 30° , 45° 的余弦函数值:

$$\text{COS} (3.1416 / 180 * 30) = 0.8661$$

$$\text{COS} (3.1416 / 180 * 45) = 0.7071$$

6. 反正切函数: ARCTAN (x)

其中 x 为整数或实数，反正切函数值一律为实数，也就是弧度数。例如：

$$\text{ARCTAN}(1.7321) = 1.0472 \quad \text{即 } 60^\circ$$

$$\text{ARCTAN}(-1) = -0.7854 \quad \text{即 } -45^\circ$$

7. 指数函数：EXP (x)

EXP 取自 EXPONENT (指数) 的缩写。

其中 x 为整数或实数，指数函数值一律为正实数。例如：

$$\text{EXP}(2) = 7.3890$$

$$\text{EXP}(-1.0) = 0.3670$$

此外，OMSI PASCAL 语言补充规定了以10为底的指数函数：EXP 10 (x)，其用法基本上与EXP (x) 相同。

8. 自然对数函数：LN (x)

LN 取自 NATURAL LOGARITHM (自然对数) 的缩写。

其中 x 为正整数或正实数，函数值一律为实数。负数不能求其自然对数值。例如：

$$\text{LN}(289.2) = 5.6672$$

LN (-100) 在程序运行时出错。

此外，OMSI PASCAL 语言补充规定了以10为底的对数函数：LOG (x)。其用法类似于LN (x)。

二、逻辑判断函数 这一类函数是指能够产生布尔类型数据结果的函数，共有三种标准函数：

1. 奇数函数：ODD (x)

其中 x 必须为整数，如果 x 为奇数值，那末函数值为 TRUE (真)，否则函数值为 FALSE (假)。例如：

ODD (-101) 为 TRUE

ODD (+256) 为 FALSE

2. 行结束函数：EOLN (x)

EOLN 取自 END OF LINE (行结束) 的缩写。

其中 x 必须为文件类型的变量。在读文件时，如果读到“本行结束的字符”时，则函数值为 TRUE，否则为 FALSE。

在OMSI PASCAL 版本中，规定标准函数EOLN (x) 只适用于 TEXT 文件。

3. 文件结束函数：EOF (x)

EOF 取自 END OF FILE (文件结束) 的缩写。其中 x 必须为文件类型的变量。在读文件时，若读到“文件结束的字符”时，EOF (x) 为 TRUE，否则为 FALSE。

在OMSI PASCAL 版本中，规定EOF (x) 只适用于 TEXT 文件。

关于 EOF (x) 和 EOLN (x) 的问题，本书第九章文件中有进一步的说明。

三、转换函数 转换函数就是指能够对数据类型进行转换的函数，标准函数中有四种属于这一类：

1. 截尾函数：TRUNC (x)

TRUNC 取自 TRUNCATION (截尾) 的缩写。

其中 x 必须是实数类型数据。其函数值是无条件地截去实数的全部小数部分，取其整数部分。因此，函数值是整数。例如：

TRUNC (+6.3) = 6

TRUNC (3.7) = 3

TRUNC (-7.9) = -7

2. 舍入函数: ROUND (x)

其中 x 必须是实数类型数据。其函数值是将实数的整个小数部分进行四舍五入, 然后取其整数部分。因此函数值是整数。

$$\text{ROUND} (x) = \begin{cases} \text{TRUNC} (x + 0.5) & x \geq 0 \\ \text{TRUNC} (x - 0.5) & x < 0 \end{cases}$$

例如:

ROUND (125.7) = 126

ROUND (318.4) = 318

ROUND (-5.7) = -6

关于截尾函数和舍入函数, 有两点注意。一是这两种函数都没有改变数据的正负性质; 二是在使用这两种函数时要防止超过整数范围。在 PDP-11/03 计算机系统上, 下述例题就无法进行:

TRUNC (327689.536)

ROUND (32767.8)

因为均已超出允许的整数范围。

3. 序数函数: ORD (x)

ORD 取其 ORDER (序号) 的缩写。

其中 x 一般为字符类型数据。函数值为该字符所对应的字符码。因此函数值一定是整数类型数据, 而且是范围很有限的整数。在 PDP-11/03 计算机系统里, 函数值是从 32 至 126。

例如:

ORD ('A') = 65

ORD ('5') = 53

ORD ('~') = 126

此外, x 也可以为布尔类型数据。例如:

ORD (TRUE) = 1

ORD (FALSE) = 0

在第六章用户自定义型数据中, 还会讨论自变量 x 还可以为枚举类型数据。

4. 字符函数: CHR (x)

CHR 取自 CHARACTER (字符) 的缩写。

其中 x 必须为一定范围的整数类型数据。函数值为该整数(作为字符码)所对应的字符。如果整数对应的是控制字符或超出 ASCII 码的字符码范围, 那末就不存在对应的字符。例如:

CHR (65) = 'A'

CHR (32) = ' ' (空格)

CHR (0) = "" 表示不存在可以打印的字符。

CHR (457) = "" 表示不存在可以打印的字符。

从上面的例题可以看出, 求序号函数和求字符函数互为逆函数。因此,

$\text{CHR}(\text{ORD}('x')) = 'x'$

$\text{ORD}(\text{CHR}(y)) = y$

四、进退函数 进退函数就是指可以按照一定的规律寻找排在该数据前面或后面的数据的函数，十七种标准函数中有两个属于这一类：

1. 前导函数： $\text{PRED}(x)$

PRED 取自 PREDECESSOR (前导) 的缩写。

其中 x 允许是整数类型、字符类型或布尔类型数据。函数值为排在该数据前面的那个数据值。 x 不能为实数类型数据。例如：

$\text{PRED}(10) = 9$

$\text{PRED}(-5) = -6$

$\text{PRED}('B') = 'A'$

$\text{PRED}(':') = '9'$

$\text{PRED}(\text{TRUE}) = \text{FALSE}$

$\text{PRED}(123.45)$ 在程序运行时会出错。

如果 x 为数据序列中的第一个数据，那么就找不到前导值。在 PDP-11/03 计算机系统里，下列函数找不到前导值：

$\text{PRED}(-32768)$

$\text{PRED}('_\text{L}')$

$\text{PRED}(\text{FALSE})$

2. 后续函数： $\text{SUCC}(x)$

SUCC 取自 SUCCESSOR (后续) 的缩写。

其中 x 允许为整数类型、字符类型或布尔类型数据。函数值为排在该数据后面的那一个数据值。 x 不能为实数类型数据。例如：

$\text{SUCC}(9) = 10$

$\text{SUCC}(-6) = -5$

$\text{SUCC}('A') = 'B'$

$\text{SUCC}(\text{FALSE}) = \text{TRUE}$

$\text{SUCC}(10.5)$ 在程序运行时会出错。

如果 x 为数据序列中的终止值，即最末一个数据，那么就找不到后续值。

例如，在 PDP-11/03 计算机系统里，下列函数：

$\text{SUCC}(32767)$

$\text{SUCC}('_\sim')$

$\text{SUCC}(\text{TRUE})$ 均找不到后续值。

由此可知，进退函数要求自变量属于有序的数据类型。整数类型，字符类型和布尔类型都是属于有序的。枚举类型数据也是有序的，同样可以作为进退函数的自变量。有序的数据类型必然有界。因此，“排头”找不到前导值，“排尾”找不到后续值。

此外，从上面例题中可以发现，前导函数和后续函数也互为逆函数。因此：

$\text{PRED}(\text{SUCC}(x)) = x$

$\text{SUCC}(\text{PRED}(x)) = x$ 。

进一步分析会发现，前导函数和后续函数的自变量为字符类型时，该两函数之间有如下

关系:

$$\begin{cases} \text{PRED}(x) = \text{CHR}(\text{ORD}(x) - 1) \\ \text{SUCC}(x) = \text{CHR}(\text{ORD}(x) + 1) \end{cases}$$

请注意, 标准函数的自变量是个非常重要的概念。它不仅可以是变量, 而且也可以是常量、表达式, 甚至允许是个函数。例如, 下列各种函数:

$\text{SIN}(x)$;

$\text{SIN}(15)$;

$\text{SIN}(x+15)$;

$\text{SIN}(\text{SIN}(x+15))$;

它们的表达都是合法的。

标准函数自变量的类型, 必须符合严格的规定。尤其是数字字符('0'..'9')与整数(0..9)的概念是完全不同的。数字字符的序号值不等于对应的数字。例如:

$\text{ORD}('6') = 54$ 而不为 6;

$\text{ORD}('0') = 48$ 而不为 0;

但是, 如果 x 为 0 至 9 之间的某一个整数, 那么下列关系:

$\text{ORD}('x') - \text{ORD}('0') = x$ 是正确的。

例如 $\text{ORD}('6') - \text{ORD}('0') = 6$;

$\text{ORD}('0') - \text{ORD}('0') = 0$ 。

本章小结

本章的重点是介绍四种标准数据类型。对于每一种数据类型, 必须掌握以下三点:

1. 类型的标志;
2. 数据的范围;
3. 数据的表示方法。

其中, 在程序设计时, 特别要注意整数的范围和实数的两种表示方法。

本章还介绍了十七种标准函数。对于每一种标准函数, 要注意弄清以下三点:

1. 函数的标志;
2. 这种函数对自变量数据类型的要求;
3. 这种函数的函数值属于哪一种数据类型。

其中, 三角函数的自变量必须用弧度表示, 不允许用角度数代入, 求字符函数与求序号函数关系很密切, 要掌握其逆函数的关系; 转换函数要防止出界; 进退函数要受到数据序列头尾的限制。

本章习题

1. 下列数据哪些是整数? 哪些是实数? 哪些超过了 PDP-11/03 计算机系统允许的范围? 哪些是非法的?

256	2.541	25 E +6	3.75 E 6	0.15 E -6
1.2 E 70	0.0	0	-678	1 E -15

2E5	E8	E-3	4321	4.375
5.5E-6.6	58793	32768	-32768	0.05E-39

2. 下列符号哪些表示字符? 哪些表示“字符串”? 有哪些不属于上述两种范围?

'A'	'!'	'9'	'78'	''''	'?'
'_'	'}'	'y'	9	'''	C
0]	STAND	'THIS IS A BOY'		

3. 下列表达式中, 哪些结果为 TRUE? 哪些结果为 FALSE?

15 > 8	125 <= 125	-300 = 300
ODD (-118)	ORD ('A') = 65	PRED (TRUE) = FALSE
SUCC ('M') = 'L'		

4. 下列函数式中, 哪些是 PASCAL 语言的标准函数? 这些标准函数的表达式中, 哪些是正确的? 哪些是错误的?

ABS (-125)	COS (15)	SIN (30)	SH (20)
5 ↑ 3	15 * * 3	TRUNC (1.5E-6)	
ORD (72.8)	CHR (65)	ODD (37.4)	
SINX	PRED ('15')	SUCC ('1')	

5. 写出下面表达式类型, 如果能确定其类型请求出其值。

- (1) SQR (9)
- (2) SQR (9.0)
- (3) SQRT (9)
- (4) SQRT (9.0)
- (5) SIN (30)
- (6) COS (45)
- (7) ORD ('y') - ORD ('a')
- (8) TRUNC (1.5E-6)
- (9) TRUNC (1.5E-6) + ROUND (-99.9)
- (10) ROUND (15.5E-1) - TRUNC (15.5E-1)
- (11) ODD (101)
- (12) SUCC (15) + PRED (1)
- (13) CHR (PRED (70))
- (14) ORD (SUCC ('?'))
- (15) ABS (-101) + ABS (101)

第三章 简单的程序设计

在第一、二章，介绍了 PASCAL 语言的程序结构，初步解决了程序设计的“骨架”问题；又介绍了基本符号，讨论了四种标准数据类型，分析了十七种标准函数，部分地解决了程序设计的“材料”问题。本章就是以此为基础，进行简单的程序设计。

本章主要讨论三个问题：

第一，PASCAL 语言中，有哪些基本运算；

第二，PASCAL 语言里的常量和变量的说明；

第三，PASCAL 语言的最基本的三种语句——读语句，赋值语句和写语句。

§1. PASCAL语言的基本运算

PASCAL 语言有着广泛的运算功能。所谓运算，就是通过某些运算符号将若干个（可以是一个）操作数组成一个表达式，产生一定的结果。基本运算最基本的格式如图3.1-1。

其中，操作数必须属于某种确定的数据类型。一般情况下，操作数 1 和操作数 2 的数据类型相同。

个别运算可以有特殊的要求。运算符决定运算的性质。

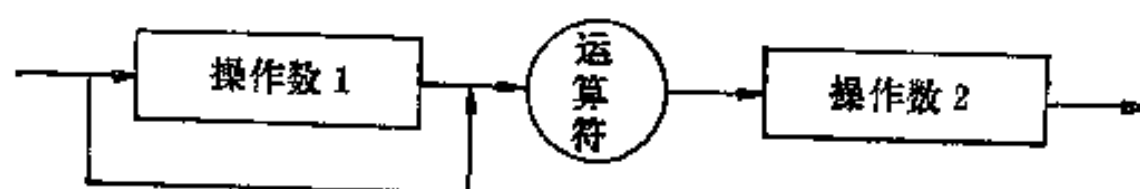


图 3.1-1 基本运算最基本的格式

PASCAL 语言的基本运算有五种：算术运算，关系运算，逻辑运算，集合运算和赋值运算。在这一节里介绍四种标准数据类型的算术运算，关系运算和逻辑运算。关于集合运算在第七章集合一节中讨论。赋值运算执行的是赋值语句的功能，在本章第四节里专门讨论。PASCAL 语言的基本运算一览表列出如下：

基本运算一览表

	运 算 符	操 作 数 类 型	结 果 类 型
算 术 运 算	+, -, *	整 数 或 实 数	整 数 或 实 数
	/	整 数 或 实 数	实 数
	DIV, MOD	整 数	整 数
关 系 运 算	=, <>	标准类型, 集合, 指针	布 尔
	<, >	标 准 类 型	
	<=, >=	标准类型, 集合	
	IN	标准类型和集合	

续上表

	运 算 符	操 作 数 类 型	结 果 类 型
逻辑运算	AND, OR, NOT	布 尔	布 尔
集合运算	+, -, *	集 合	集 合
赋值运算	=	除文件类型外各种数据类型	除文件类型外各种数据类型

一、算术运算 算术运算就是对两个（或两个以上）操作数与算术运算符组成的算术表达式进行的计算处理。算术运算共有六种：加（+），减（-），乘（*），实数除（/），整数除（DIV），取模（MOD）。操作数只能是整数型或实数型数据，运算结果或是整数型，或是实数型，它取决于操作数的类型和运算符。

加（+），减（-）和乘（*）三种是最常用的算术运算。如果两个操作数都为整数，结果是整数型数据；两个操作数都为实数时，结果也为实数型数据。例如：

$$\begin{aligned} 13 + 5 &= 18; & 13.0 - 5.0 &= 8.0; \\ 13.0 * 5.0 &= 65.0; & 13 * 5 &= 65. \end{aligned}$$

但是，这三种算术运算符中，如果两个操作数类型不同（即一个为整数型，另一个为实数型），运算结果为实数。例如：

$$13 + 5.0 = 18.0; \quad 13 * 5.0 = 65.0.$$

实数除（/）运算时，无论两个操作数是整数或实数，结果一律为实数型数据。例如：

$$13/5 = 13.0/5 = 13/5.0 = 13.0/5.0 = 2.6$$

整数除（DIV）和取模（MOD）两种算术运算，要求操作数必须为整数，这时运算结果也是整数型数据。DIV 是两整数相除，取商数的整数部分，舍去商数的小数部分。MOD 是在两个整数整除的基础上，取其余数部分，即：

$$X \text{ MOD } Y = X - (X \text{ DIV } Y) * Y.$$

$$\text{例如, } 13 \text{ DIV } 5 = 2$$

$$13 \text{ MOD } 5 = 3$$

$$\begin{aligned} 130 \text{ MOD } 50 &= 130 - (130 \text{ DIV } 50) * 50 \\ &= 130 - 2 * 50 = 30 \end{aligned}$$

DIV 和 MOD 运算，不允许操作数为实数。例如 13.0 DIV 5 或 43 MOD 5.0 都是非法的。

二、关系运算 关系运算就是对两个（或两个以上）操作数与关系运算符组成的布尔表达式进行的比较与判断处理。

操作数可以是任何一种标准数据类型，但必须是同一类型。否则，无法进行比较和判断处理。关系运算的结果一定是布尔类型数据。

常用的关系运算符有六种：=, <, >, <=, >=。

下面举几个例子说明：

$$13 > 5 \quad \text{为 TRUE};$$

$$14.5 <= 2.7 \quad \text{为 FALSE};$$

$$37 >= 89 \quad \text{为 FALSE};$$

'A'='B' 为FALSE。

'>' > '<' 为TRUE；

TRUE< > FALSE 为 TRUE。

此外，集合类型数据中有=，< >，>=，<= 四种关系运算符，还有一种特殊运算符IN，关于集合的这五种关系运算，在第七章中予以讨论。

三、逻辑运算 逻辑运算就是对若干个（也可以一个）操作数通过逻辑运算符组成的布尔表达式进行的逻辑运算处理。

其中操作数必须是布尔型数据。运算的结果也一定是布尔类型数据。

逻辑运算符共有三个：NOT，AND 和OR。NOT，非的意思，也可称“反”，AND，与的意思。OR，或的意思。在逻辑电路中，它们分别代表非门，与门和或门。

AND 和 OR，要求两个（或两个以上）操作数，而 NOT 只要一个操作数即可。

逻辑运算举例说明如下：

TRUE AND FALSE 为 FALSE；

TRUE OR FALSE 为 TRUE；

NOT FALSE 为 TRUE。

在上面的所有例题中，操作数都是常量。实际的程序中，操作数允许是常量，变量，因子和项，甚至是一个复杂的表达式。其中所说的“因子”，可以是常量，变量和函数等；而所谓“项”，可以由因子通过*，/，DIV，MOD 和 AND 等组成。简单表达式由项通过+，-，OR 等组成。复杂表达式可以由简单表达式通过关系运算符构成。关于变量，因子，项及表达式的问题请参阅附录中的语法图。

在进行基本运算时，要注意以下三点：

1. 在数学上允许书写如下表达式：

$A \geq B \geq C \geq D$

但是，在 PASCAL 语言里，必须写成如下形式：

$(A \geq B) \text{ AND } (B \geq C) \text{ AND } (C \geq D)$

2. 在逻辑运算时，操作数与逻辑运算符之间，必须留一空格。例如：

XANDY 是错误的形式；

X AND Y 是正确的形式。

如果操作数本身是一个布尔表达式，则必须用圆括号（ ）将其括起来，与逻辑运算符之间最好留一空格。当然，这时也允许不留空格。例如：

$(A \geq B) \text{ AND } (C = D) \text{ OR } (\text{NOT } E)$

这种表达不仅正确，而且比较清晰。

$(A \geq B) \text{ AND } (C = D) \text{ OR } (\text{NOT } E)$

这种表达也是正确的，允许这样写，但看起来不清晰。

$A \geq B \text{ AND } C = D \text{ OR } \text{NOT } E$

这种表达则是错误的，不允许的。

3. 执行基本运算，其优先级如下：

(1) 圆括号（…（…（…）…）），按照由内到外，逐层展开的规律进行；

(2) 逻辑非运算：NOT；

(3) 乘除类运算：AND，*，/，DIV，MOD；

- (4) 加减乘除运算: OR, +, -;
- (5) 关系运算: =, <, >, <=, >=;
- (6) 在同一级, 按照自左至右的顺序依次执行。

§2. 常量定义和变量说明

在 PASCAL 语言中, 凡是在程序执行部分用到的常量和变量, 都必须在程序说明部分加以定义和说明。

一、常量定义 在第一章中已经介绍过, PASCAL 语言定义了三个常量: FALSE, TRUE, MAXINT。其中 MAXINT 是机器的最大整数, 它的大小与具体的计算机有关。

显然, 仅仅有这三个常量远远不能满足程序设计的需要。为了书写方便, 名称形象化, PASCAL 语言允许用户自己定义一些标识符, 用来代表一些常量。常量定义的格式如图 3.2-1

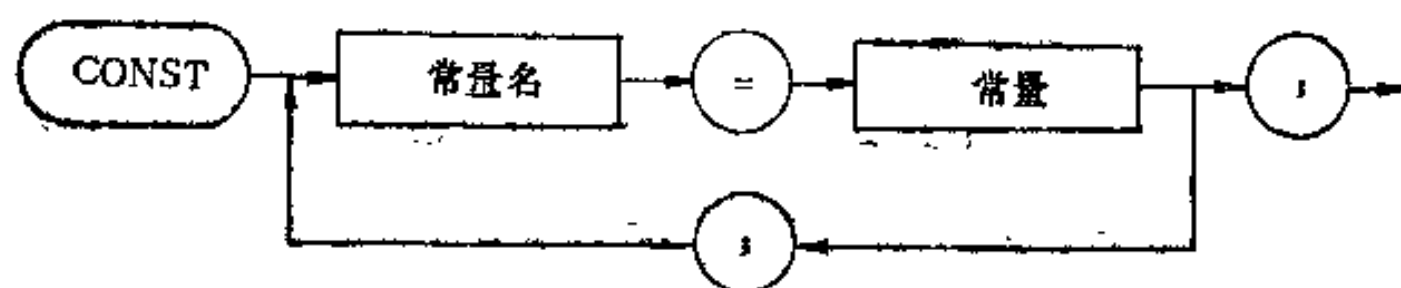


图 3.2-1 常量定义的格式

其中:

CONST 为常量定义的保留关键字。

常量名由用户自己定义, 但必须符合用户标识符的规定。常量可为任何一种标准数据类型的数据。

常量名与常量必须一一对应, 否则将造成混乱和错误。一个程序中可以定义若干个常量, 共用一个 CONST, 也允许每个常量前后加上 CONST。例如:

CONST

MAXNUMBER=10000;

MINNUMBER=50;

PI=3.1415925;

BLANK=' ';

T=TRUE;

F=FALSE;

上述常量定义是合法的。

下面的常量定义:

CONST

A=100;

B, C=60;

D=100, 150;

显然是错误的, B 和 C 都是常量 60, 不能写成 B, C=60; D 代表常量 100, 又代表常

量150, 这是不允许的。常量定义必须是单值性的。

OMSI PASCAL 版本, 对常量定义有扩充。它允许对整数常量用八进制定义, 办法是在常量后面加一个字母B, 例如:

CONST

A=35B;

它表示常量标识符A代表八进制整数35, 即十进制整数29。

二、变量说明 任何一个有使用价值的程序, 离不开设置变量和作变量说明。在初次与变量打交道的读者, 或许要问什么是变量, 有无变量名称? 变量属于哪一种数据类型? 变量的内容如何确定? 变量的内容如何输出? 前面两个问题, 在PASCAL语言中, 是通过变量说明解决的。后两个问题在下面的几节里进行讨论。

PASCAL语言规定, 变量说明的格式如图3.2-2;

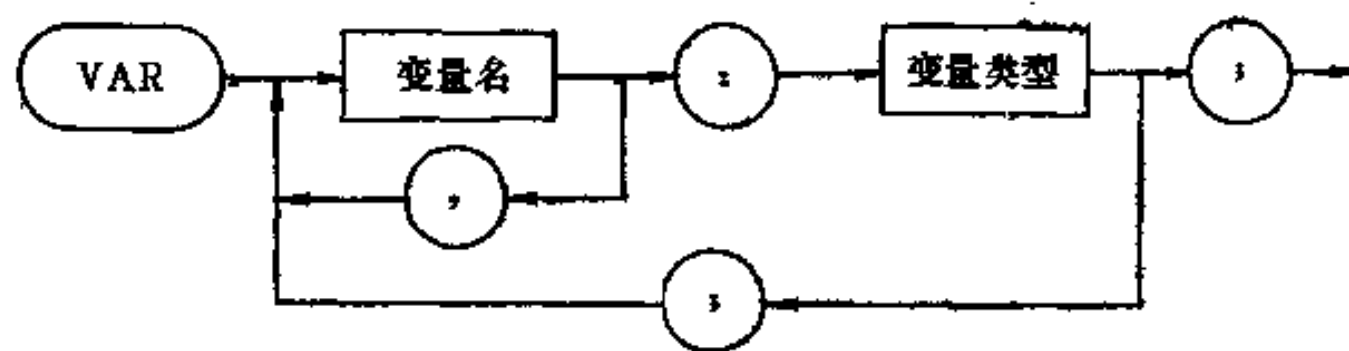


图 3.2-2 变量说明的格式

其中, VAR 是 PASCAL 语言本身选用作“变量说明”的保留关键字。

变量, 顾名思义, 就是指程序运行过程中某些可以在一定范围内, 按照一定的规律或指定的方式不断改变其内容即数据值的量。每个变量自然要有一个名称, 这就是变量名。变量名由用户自己定义, 但必须符合标识符的规定。

变量类型, 就是回答变量属于哪一种数据类型的问题, 它可以是任何一种数据类型。

在一个程序内部, 一个变量只能属于一种确定的数据类型。

一个程序中, 可以定义许多不同的变量, 共用一个 VAR, 也允许在每个变量说明前加上 VAR, 例如:

VAR

I:INTEGER;

ROOT1, ROOT2:REAL;

CH1, CH2, CH3:CHAR;

OVERFLOW:BOOLEAN;

其中 INTEGER, REAL, CHAR 和 BOOLEAN 分别为 PASCAL 语言本身选用来说明整数类型, 实数类型, 字符类型和布尔类型的标准标识符。

下列变量说明:

VARIABLE

I, J, K:INTEGER;

R1, R2:CHARACTER;

R2, B:BOOLEAN;

显然, 这是错误的。错误有三处:

第一, 变量说明的标识符用错了。它只能是 VAR, 不能用 VARIABLE。

第二，字符类型的标识符用错了。它应该是 CHAR，不能用 CHARACTER

第三，变量 R2 的类型有二义性，它不能既为字符类型，又是布尔类型。

§3. PASCAL 的语句类型

前面介绍了四种标准的数据类型，常量定义和变量说明，这些属于程序设计的任务之一——“对数据进行描述”。程序设计的另一项任务是“对数据进行操作”。

在汇编语言中，对数据进行操作是通过指令（INSTRUCTION）来实现的。如指令 ADD, SUB, JSR 等。

在高级语言，比如 PASCAL 语言，对数据进行操作是通过语句（STATEMENT）来实现的。如语句 READ, WRITE 等。

从本质上讲，语句实际上是由一系列指令组成的，指令是由一系列机器代码组成的，而机器代码则是由计算机电位的高低或脉冲的有无来实现的。电子计算机就是按照程序设计的要求，通过执行一系列指令，自动完成用户交给的任务。

PASCAL 语言的通用语句，按照语句标号的有无，可分为标号语句和无标号语句。所谓标号语句是指语句前面有一个语句标号作为前缀，该语句标号为转移语句所引用。无标号语句按其结构又可分成两大类：基本语句和构造型语句。

基本语句结构简单。它包括赋值语句，转移语句，空语句和过程语句。过程语句有很多种，本章只讨论读语句和写语句。其它过程语句将陆续介绍。

构造型语句结构复杂。它包括复合语句，重复语句，条件语句和开域语句四类。其中重复语句又有三种：当语句，直到语句和循环语句。条件语句有两种：如果语句和情况语句。

本章只介绍最基本的语句：赋值语句，读语句和写语句。在第四章介绍标号语句和三类构造型语句。第五章介绍非标准过程语句。从第七章到第九章介绍各种标准过程语句。开域语句在第七章“记录类型”一节中讨论。

PASCAL 语言的语句分类表，列表如下：

PASCAL 语言的语句分类表

无标号语句	基本语句	赋值语句	
		过程语句	读语句 写语句 其它过程语句
		转移语句	
		空语句	
	构造型语句	复合语句	
		重复语句	当语句 直到语句 循环语句
		条件语句	如果语句 情况语句
		开域语句	
	标号语句		

§4. 赋值语句

设置了变量之后, 如何确定变量的内容? 用赋值语句来确定变量的内容是最常用的方法。

赋值语句是最简单的语句。它的功能主要是确定变量的内容。赋值语句的格式如图3.4-1其中:

变量名必须在变量说明时已经确定;

$:=$ 是赋值运算符。它表示将运算符右侧的表达式运算结果送进运算符左侧变量所在的存贮单元;

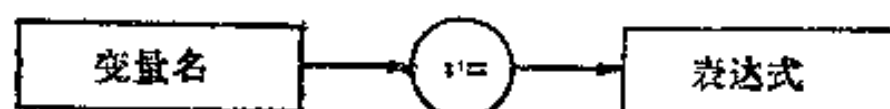


图 3.4-1 赋值语句的格式

表达式允许为常量, 变量, 函数, 算术表达式或布尔表达式。一般地说要求表达式运算的结果与变量属于同一数据类型。

例如, 下述赋值语句:

$Y := 0.5;$

$X := Y;$

$X := Y + 2.5;$

$X := \sin(Y + 3.5);$

如果在变量说明部分已经说明 X 和 Y 是实数型变量, 上述赋值语句都是合法的。

例3-1A 已知半径为0.5米, 高为10米的圆柱体, 求其体积。

分析: 由数学公式写出圆柱体的体积 V 的表达式, $V = \pi R^2 H$

其中 π 为圆周率, 它是一个常量。 R 和 H 为圆柱体的半径和高度, 不同的圆柱体, R 和 H 的值是不同的。在本例题的程序中, 首先在常量说明中定义圆周率 $PI = 3.14159$; 然后在变量说明中规定半径 R 和圆柱体体积 V 为实数类型变量, 圆柱体 H 为整数类型变量。

程序如 FIG SMP01A PROGRAM 所示。

```
PROGRAM SMP01A(INPUT OUTPUT);
CONST
PI=3.14159;
VAR
H:INTEGER;
R,V:REAL;
BEGIN
H:=10;
R:=0.5;
V:=PI*SQR(R)*H
END.
```

FIG SMP01A PROGRAM

必须指出, PASCAL 语言中的赋值语句不同于数学上相等的概念。例如, I 和 K 是整数型变量, 下述表达式:

$K := 10 - K;$

$I := I + 2;$

它们都是正确的赋值语句，分别表示10减去K单元中目前的数值再送进K单元中去；I单元目前的数值加上2以后再送进I内存单元中去。

但是，在数字上， $K = 10 - K$ 是一个方程，其解为5。而 $I = I + 2$ 则是一个无解的方程。

此外，赋值语句还常用于用户定义的函数赋值。即将某一个表达式的值赋给一个函数名称，这在函数说明的执行部分是不可缺少的语句。其格式为图3.4-2。



图 3.4-2 用户定义函数中的赋值语句格式

其中函数名由用户自己定义，但必须符合标识符的有关规定。这个问题在第五章过程和函数中讨论。

在一般情况下，要求变量（或函数）与赋值运算符右侧的表达式的结果具有相同的数据类型。但是，允许有两个特例除外：

1. 变量（或函数）是实数类型，表达式运算的结果为整数类型或由整数构成的子界类型；
2. 变量类型是表达式类型的子界，或表达式类型是变量类型的子界。

关于子界的问题，将在第六章的“子界类型”一节中讨论。

要知道，在PASCAL语言中，赋值运算符“:=”与等号“=”明显地区别开来，这是一个优点。这样可使用户在进行比较判断与赋值运算时不会发生混淆。

§5. 输 入 及 读 语 句

在例3-1 A中，通过赋值语句确定变量R和H的内容。这种方式要求用户在程序设计阶段就确定变量的具体内容或固定的表达式。如果要求这些变量的内容由用户在程序运行阶段灵活地确定，用赋值语句就不方便了。这样，要求程序中有输入数据的功能。能够输入数据的语句是读语句。

读语句是确定变量内容的又一种方法，而且是一种更灵活的方法。它有两种形式，其格式如图3.5-1

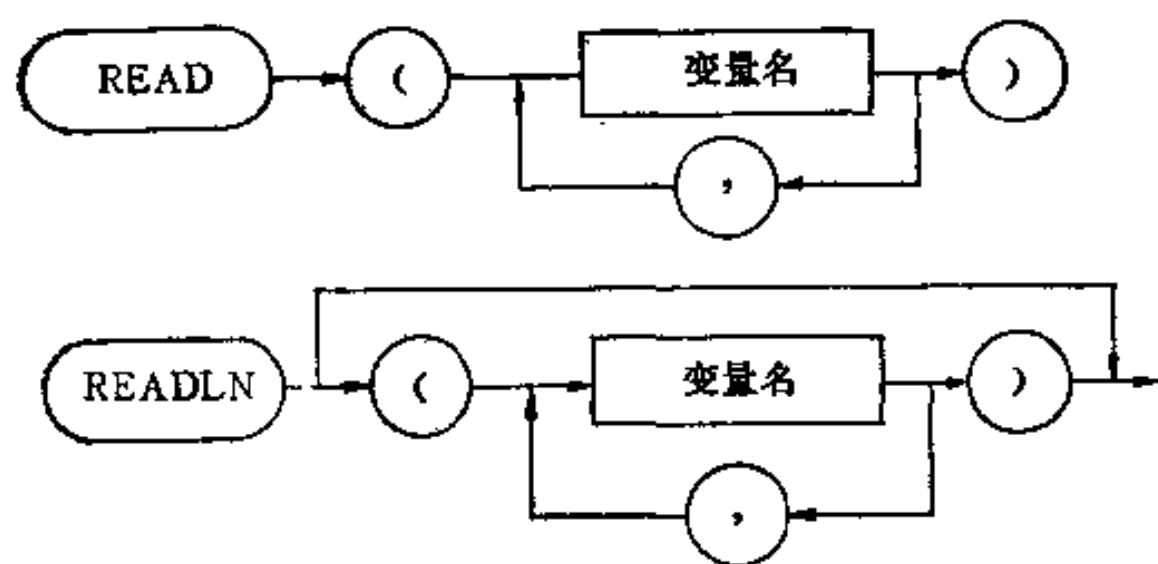


图 3.5-1 读语句的两种格式

式如图3.5-1

其中：

READ和READLN是读语句中所用的标准过程名。它们是PASCAL语言的标准标识符。

变量名必须在程序说明中预先说明，它的值可以属于整数类型、实数类型或字符类型，不能是布尔类型。

在PASCAL语言中，READ语句和READLN语句的功能基本相同。它们可以用在程序中需要输入数据的任何位置，而且可以通过一个读语句，一次读入若干个变量的值。

READ语句和READLN语句的主要区别有以下两点：

1. READ语句仅仅要求一个一个数据不断输入，并不要求换行。如果本行还有其它数

据，下一个 READ 语句可以接着使用。READLN 语句，不仅要求一个一个数据接着输入，一旦本语句读完所要求的数据，则不管本行还剩多少内容，都要跳到下一行去。也就是说，如果还有下一个读语句的话，它要从另外一行中读取数据，原来一行中剩余的内容不能再读取了。

2. READ 语句必须至少输入一个数据。而 READLN 语句允许不输入任何数据，只是执行换行要求。例如，在程序中：

```
READLN;
```

```
READ;
```

前者是合法的语句，后者则是不允许的。

如果一个程序段中只有一个读语句，READ 语句和 READLN 语句作用完全相同。如果超过一个读语句，则要注意有无换行的要求。在通过终端键盘输入数据时，更应慎重对待。本章中涉及的输入数据，基本上是通过键盘输入的。请注意，这时两个数据之间（指整数或实数）用一个或几个空格符隔开，也可以用逗号隔开。

例3-1B 通过终端键盘输入一个圆柱体的半径R和高度H，计算这个圆柱体的体积。

程序如 FIG SMP01B PROGRAM 所示。

```
PROGRAM SMP01B(INPUT,OUTPUT),
CONST
  PI=3.14159;
VAR
  H:INTEGER;
  R,V:REAL;
BEGIN
  READLN(H);
  READLN(R);
  V:=PI * SQR(R) * H
END.
INPUT,
10
0.5
FIG SMP01B PROGRAM
```

在这个程序中用了两个 READLN 语句。在程序的下部“INPUT:”表示下面的数据是通过终端输入的：10

0.5

表示输入高度H的值10之后，换行，再输入半径R为0.5，这样数据H和R的数值才能读入机器之中。

在这个程序中，R被定义为实数类型变量。如果实际输入一个整数，同样是允许的。机器将自动地将它转换为实数参加运算。比如半径R为2。用户只要输入2，不必输入2.0。但在计算机中因为R是实数，所以它是以实数2.0存在两个存贮单元中。如果R是整数，输入2，那末整数2是存贮在一个存贮单元中的。

举几个例题来说明读语句的应用。假定，A，B，C和D都是整数型变量，已在变量说明中予以说明过了。执行部分中有下列语句：

1. READ(A);

终端键盘输入: 10↵

执行结果: A=10

注意: “↵”代表回车。在输入数据10后, 不按回车键, 程序并不执行操作, 按下回车键后, 程序才真正开始执行。在PDP-11/03计算机系统上, 回车键 (RETURN) 包含两个功能: 回车 (CARRIAGE RETURN) 和换行 (LINE FEED)。执行后 整数送进单元 A, 即单元 A 的内容为10, 表示 A=10。

2. READ (A, B, C, D);

输入数据分为四种情况, 其执行结果各不相同。

(1) 终端键盘输入: 10, 20, 30, 40↵

执行结果: A=10; B=20; C=30; D=40。

(2) 终端键盘输入: 10, 20, 30, 40, 50↵

执行结果: A=10; B=20; C=30; D=40;

而数据50是多余的, 没有赋给任何一个变量。

(3) 终端键盘输入: 10, 20, 30↵

执行结果: A=10; B=20; C=30;

因为只输入三个数值, 故 D 没有被赋值, 程序等待用户继续输入数据。

(4) 终端键盘输入: 10, 20↵

30, 40↵

执行结果: A=10; B=20; C=30; D=40。

由此可知, 在执行读语句时, 出现如下两种情况是允许的:

(1) 如果输入的数据个数多于变量的个数, 在语法上是合法的。这时后面的数据是多余的, 不起任何作用;

(2) 如果输入的数据个数少于变量的个数, 那么在语法上也是合法的。这时后面的变量没有赋值, 维持原状, 等待用户继续输入数据。

3. 下列读语句之间有功能相同的关系:

READ(A); READ (B, C, D) 与 READ (A, B, C, D);

READ(A); READLN; 与 READLN(A);

READ(A, B); READLN(C, D); 与 READLN(A, B, C, D);

4. 下列读语句之间功能不同:

READLN(A); READLN(B); 与 READLN(A, B);

前者要求输入 A 以后必须换行, 然后才能输入 B; 后者则是输入 A 和输入 B 以后再换行。

READLN; READLN(A); 与 READLN(A);

两者功能也不相同, 前者要先换行, 再输入数 A, 最后换行; 后者要求先输入数 A, 然后换行。

5. READLN(A, B); READLN(C, D);

(1) 终端键盘输入: 10, 20↵

30, 40↵

执行结果: A=10; B=20;

C=30; D=40;

(2) 终端键盘输入: 10, 20, 30, 40✓

执行结果: A=10; B=20;

其中输入的30和40两个数是多余的, 没有赋值给任何变量。C和D没有被赋值, 等待用户继续输入。

(3) 终端键盘输入: 10, 20, 30✓

40, 50, 60✓

执行结果: A=10; B=20;

C=40; D=50;

其中输入的30和60两个数是多余的, 没有赋给任何变量。

(4) 终端键盘输入: 10✓

20✓

30✓

40✓

执行结果: A=10; B=20;

C=30; D=40;

总而言之, 在执行 READLN 语句时, 其基本规律与执行 READ 语句大体相同: 不同的是, 后者不存在换行问题, 而前者必须注意换行的问题。这就是说, 要将输入数据的个数与换行前变量的个数相比较, 如果输入的数据多了, 多余的数据不起作用, 如果输入的数据比变量少, 则等待输入。

§6. 输出及写语句

在例3-1A和例3-1B中, 尽管通过赋值语句或读语句输入了必需的数据。但是, 计算机计算的结果如何, 即圆柱体的体积是多少, 用户并不知道。为了得到程序运行阶段产生的中间数据, 以及最后结果, 程序中必须有输出数据的功能。能够输出数据的语句是写语句。

写语句是输出数据的一种语句。在有价值的程序中, 几乎都有写语句。它有两种形式, 其格式如图3.6-1

其中:

WRITE 和 WRITELN 是写语句中所用的标准过程名, 是 PASCAL 语言的标准标识符;

项目名可以是常量、变量或函数的名称, 也可以是字符串或表达式。

如果项目名是常量名, 则直接输出该常量的值; 如果是变量名,

则输出该变量的内容。写语句中, 变量允许属于任何一种标准数据类型。如果项目名是字符串, 那末在输出设备上重现字符串的内容。这给输出各种表格带来很大的方便。如果项目名是函数或表达式, 则首先对函数或表达式进行运算, 然后输出运算的结果。

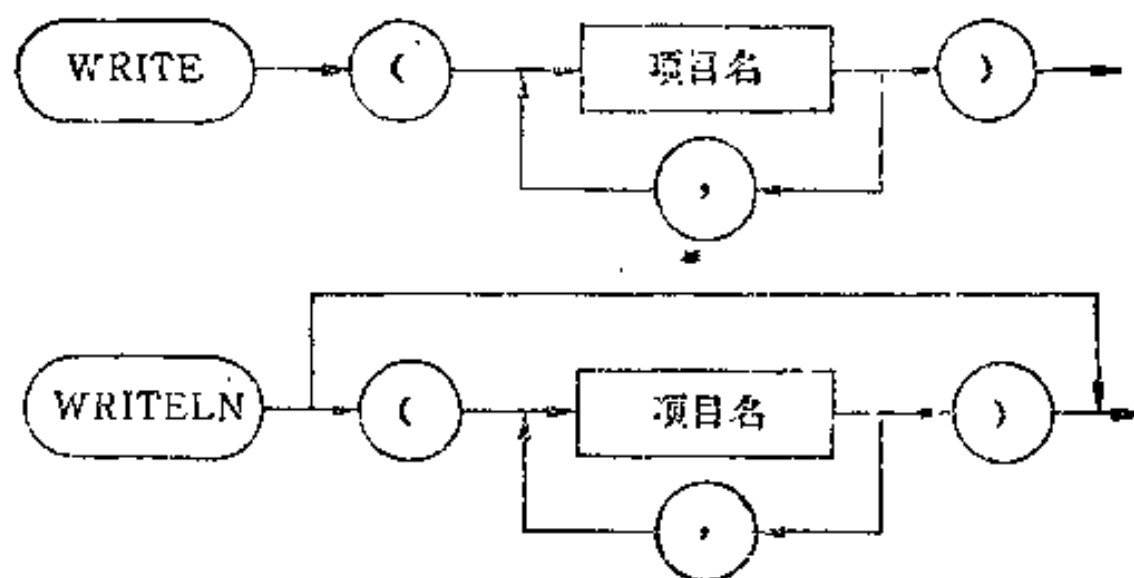


图 3.6-1 写语句的格式

在 PASCAL 语言中, WRITE 语句和 WRITELN 语句的功能基本相同。它们可以在一个程序中的任何要进行输出操作的位置,而且可以在一个写语句中,输出若干个项目。

两种格式的写语句的主要区别有以下两条:

第一, WRITE 语句是一项接着一项地输出,但不换行; WRITELN 语句是一项接着一项地输出,输出完最后一项后自动换行。

第二, WRITE 语句至少必须输出一项内容,而 WRITELN 语句允许不输出任何内容,仅仅换行。因此,在下述程序段中有以下语句:

⋮

WRITELN;

WRITE;

前者 WRITELN; 是合法的语句,后者则是不允许的。

例3-1C 输入一个圆柱体的半径和高,计算它的体积,最后输出计算结果。

分析:

基本与例3-1B相同,仅仅最后通过写语句输出体积V。

程序如 FIG SMP01C PROGRAM 所示。

```
PROGRAM SMP01C(INPUT,OUTPUT);
CONST
  PI=3.14159;
VAR
  H:INTEGER;
  R,V:REAL;
BEGIN
  READLN(H);
  READLN(R);
  V:=PI*SQR(R)*H;
  WRITELN('V=',V)
END.
INPUT,
10
0.5
OUTPUT,
V= 7.853750E+00
FIG SMP01C PROGRAM
```

其中,写语句有两项,第一项是字符串'V=',第二项是变量名V。这样做是为了增加输出数据的可读性。

在程序下部的“INPUT,”以下的字符,表示输入的数据,通常通过键盘打入。“OUTPUT,”以下的内容,表示输出的数据,在输出设备(如终端屏幕,或行打印机)上显示。

为什么输出的数共占13列,首列为空格?为什么不是十进制表示?为什么E前而有七位有效数字?这都是由实数型变量输出的格式所决定的。

在 PASCAL 语言中,输出数据的格式非常重要。在进行数据处理,制订各种表格时尤其要注意这一点。

每一种类型数据所占的列数称为场宽 (FIELD WIDTH)。PASCAL 语言对各种数据定义了标准场宽。标准场宽受计算机系统的约束。其规定如下:

数据类型	场宽	
	标准PASCAL	OMSI—PASCAL
整型	10	13
实型	22	13
布尔型	10	6
字符型	1	1
ALFA 型	10	10
字符串	字符串长度	字符串长度

其中 ALFA 型数据和字符串的定义将在第七章“数组类型”一节中介绍:

场宽的大小与数据的范围及机器中的表示方法有关, 其中整数的场宽中包括符号位。显示时, 符号位在整数值的最前面, 但不一定在第一列。例如, 显示 -125 为 `-125`, 不是一 `-125`。符号位为正号时不显示。布尔型数据显示 `FALSE` 或 `TRUE`;

实数型数据场宽是按照科学表示法显示的, 在 PDP-11/03 计算机上, 它的格式如下:

f	d ₁	.	d ₂	d ₃	d ₄	d ₅	d ₆	d ₇	E	fj	j1	j2
---	----------------	---	----------------	----------------	----------------	----------------	----------------	----------------	---	----	----	----

其中, f 为实数的符号位, 符号位为正号时不显示。fj 为指数部分的阶符号位, 无论正号还是负号都显示。d₁~d₇ 为尾数部分的七位数, j₁ 和 j₂ 为指数部分的阶码数。在实数不为零的情况下, 数字 d₁ 一律不为零。

例如: 已知 R1 为 123.4567, R2 为 -0.654321, 执行语句 `WRITELN(R1)` 和 `WRITELN(R2)` 之后, 终端显示器中显示:

`1.234567E+02`

`-6.543210E-01`

为了得到所要求的输出格式, 用户也可以自己定义场宽, 一般有两种方法。

一、单场宽。格式如图 3.6-2 所示。

其中项目名允许为常量名, 变量名, 函数名, 字符串或表达式。但是, 结果为实数型的数据是不允许的。场宽一律为正整数。



图 3.6-2 单场宽格式

例如, 已知 I 为整数 125, CH 为字符 '#', B 为布尔量 TRUE。

执行下列语句:

```
WRITELN(I:6);      WRITELN(CH:6);
WRITELN(B:6);      WRITELN('END.':6);
```

程序运行后, 终端显示器显示:

`125`

`#`

└└TRUE

└└END.

二、双场宽。格式如图3.6-3:

其中项目名允许为实数型变量, 函数名或算术表达式, 它们运算的结果必须是实数型数据。这种写语句显示的是用小数形式表示的

实数。场宽1为总列数, 它包括符号位, 整数部分列数, 小数点和小数部分列数。

场宽2号表示小数部分列数。场宽1和场宽2都用正整数表示。而且要求场宽1大于场宽2。

例如, 已知R为实数-654.321。

执行语句WRITELN(R:10:4)后, 终端显示器显示如下:

└-654.3210

执行语句WRITELN(R:8:2)之后, 终端显示器显示如下:

└-654.32

这时, 最后一位的“1”不再显示了。但是, 必须指出, 这个数仍然存在, 没有舍去, 不影响进一步运算。

例3-2. 为了综合分析写语句中场宽的各种情况, 我们假定: I和D为整数型变量, R为实数型变量, B为布尔型变量, C为字符型变量。首先输出赋值之前各变量的内容。然后输入I和R, 对B和C赋值。最后, 按各种场宽输出赋值后各变量的内容。

程序如FIG SMP02A PROGRAM所示。有二点必须请读者注意:

1. INPUT: 下面的内容为用户输入的数据;
 2. OUTPUT: 下面的内容为终端显示器输出的数据。
- 它们不是程序的一部分, 而是为读者便于理解而附加的。

```
PROGRAM SMP02A(OUTPUT);
VAR
  I,D:INTEGER;          R:REAL;
  C:CHAR;               B:BOOLEAN;
BEGIN
  WRITELN('I0=',I);
  WRITELN('R0=',R);
  WRITELN('B0=',B);
  WRITELN('C0=', ''',C,'''');
  D:=ORD(C);
  WRITELN('D0=',D:3);
  D:=ORD(PRED(' '));
  WRITELN('D1=',D:3);
  READLN(I,R);
  B:=I>1000;
  C:='#';
  WRITELN('I1=',I);
```

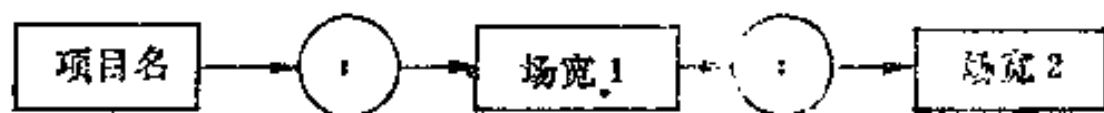


图 3.6-3 双场宽的格式


```

WRITELN('I2=',I:15);
WRITELN('I3=',I:3);
WRITELN('R1=',R);
WRITELN('R2=',R:15:5);
WRITELN('R3=',R:6:3);
WRITELN('B1=',B);
WRITELN('B2=',B:15);
WRITELN('B3=',B:3);
WRITELN('C1=',',',C,',');
WRITELN('C2=',',',C:15,',');
WRITELN('C3=',',',C:3,',');
END.
INPUT,
5678 1234.56789
OUTPUT,
I0= 0
R0= 0.000000E+00
B0= FALSE
C0=''
D0= 0
D1= 31
I1= 5678
I2= 5678
I3=5678
R1= 1.234568E+03
R2= 1234.56800
R3= 1234.568
B1= TRUE
B2= TRUE
B3= TRUE
C1=' #'
C2=' #'
C3=' #'

```

FIG SMP02A PROGRAM

现在，我们分析一下输入和输出的关系，以及场宽的有无及大小对输出的影响。

为了输出的字符比较清楚，我们在字符的两侧都加一个撇号'。为此，在语句 WRITE 中加上''，则在终端显示器上显示'。

1. 在对变量赋值之前，各种类型的变量的数据值如下：

整数型变量值 (I0)：整数零；

实数型变量值 (R0)：实数零；

布尔型变量值 (B0)：FALSE；

字符型变量值 (C0)：没有任何字符，但也不是空格字符，因为其字符的序数值(D0)为零。而空格字符的序数值应为32。空格字符前导的序数值 (D1) 为31。

2. 在没有定义场宽的情况下, 各类变量按照标准场宽的格式输出。

整数型变量值 (I1): 占13列;

实数型变量值 (R1): 占13列, 科学表示法;

布尔型变量值 (B1): 占6列;

字符型变量值 (C1): 占1列。

3. 在用户定义单场宽时, 如果场宽满足数据的需要, 那末输出格式既服从数据需要, 也满足场宽定义。采用的方法是在数据的左侧用空格字符补充, 用满足自定义场宽的要求。

整数型变量值 (I2): 占15列, 其中数据占4列, 左侧补充11列空格字符。

布尔型变量值 (B2): 占15列, 其中数据占4列, 左侧补充11列 (实际上9列) 空白字符。

字符型变量值 (C2): 占15列, 其中数据占1列, 左侧补充14列空白字符。

4. 在用户定义单场宽时, 如果场宽不能满足数据的需要, 即自定义场宽的列数小于数据所需列数。那末, 输出格式服从数据的需要而突破场宽的定义。但是, 如果数据为正整数, 其符号位自动省去, 不留位置。

整数型变量值 (I3): 占4列, 因为数据需要4列, 突破3列的限制;

布尔型变量值 (B3): 占6列, 因为数据需要6列, 突破3列的限制;

字符型变量值 (C3): 占3列, 数据只需要1列, 左侧补充两列空白字符。

5. 在用户定义双场宽时, 如果场宽满足数据的需要, 那末输出格式既服从数据的需要, 也满足场宽定义。采用的方法是: 在数据的左侧用空白符补充以满足场宽1的定义, 在数据的右侧用零补充以满足场宽2的定义。实数用十进位表示法, 有七位有效数字。如实数型变量值 (R2), 数据左侧补充5列空白字符, 右侧补充两个零。其中数字8是四舍五入的结果。

6. 在定义双场宽的情况下, 如果场宽不能满足数据的需要, 那末, 输出格式主要服从数据的需要。它突破场宽1的定义, 同时服从场宽2的定义, 自动舍去小数的后面几位; 如采数据为正实数, 符号位正号虽然不输出, 但要留出一列位置, 这一点与整数不同。如实数型变量值 (R3), 突破了场宽1 (6列), 服从了场宽2 (3列) 舍去了小数的后几位。同时保留符号位的位置。

总而言之, 未定义场宽时, 按标准场宽输出, 定义场宽后, 采取保证数据正确为主, 兼顾场宽定义为辅的原则。

OMSI PASCAL-1 语言扩充了对场宽的定义, 它规定, 在需要以八进制输出整数型数据时, 要在场宽之前加一个负号, 表示按八进制输出。

例3-2B 已知常数C为八进制35, 输入整数型变量I, 分别用八进制和十进制输出I和C。

例序如 FIG SMP02B PROGRAM 所示

```
PROGRAM SMP02B(INPUT,OUTPUT);
CONST
  C=35B;
VAR
  I:INTEGER;
BEGIN
```

```

    READLN(I);
    WRITELN('I1=', I:5);
    WRITELN('I2=', I:-5);
    WRITELN('C1=', C:5);
    WRITELN('C2=', C:5);
END.

```

INPUT,

19

OUTPUT,

I1= 19

I2= 23B

C1= 35B

C2= 29

FIG SMP02B PROGRAM

从程序的运行结果可以看出，凡是以八进制输出的整数，最后会以一个字母B作为标志。

§7. 字符的输入

OMSI PASCAL-1 语言在PDP-11/03计算机上的实践表明，通过终端键盘输入字符比较复杂，不象输入整数、实数那样简单明了。因此，将字符输入作为单独一节，作比较仔细的分析 and 讨论。

例3-3 输入一个字符，计算它的序数值，输出该字符及其序数值。

程序如FIG SMP03A PROGRAM所示。

```

PROGRAM SMP03A(INPUT, OUTPUT);
VAR
    IO: INTEGER;
    CH: CHAR;
BEGIN
    READLN(CH);
    IO := ORD(CH);
    WRITELN('CH=', CH);
    WRITELN('IO=', IO:3);
END.
INPUT,
OUTPUT,
CH= ' '
IO= 32

```

FIG SMP03A PROGRAM

按照通常的规律，在执行这个程序时，它一定会等待用户输入一个字符，然后再进行计算，并显示结果。

然而，事实上在执行这个程序时，在用户还没有输入任何字符时，显示器上已经显示出

输入字符和程序执行结果。

显示结果表明，程序执行时自动输入了一个字符，这个字符是空格字符，其序数值为32。

例3-4A 先输入一个整数，再输入一个字符，然后输出这个整数值和字符。

程序如 FIG SMP04A PROGRAM所示

```
PROGRAM SMP04A(INPUT, OUTPUT),
  VAR
    IO:INTEGER,
    CH:CHAR,
  BEGIN
    READ(IO),
    READLN(CH);
    WRITELN('IO=', IO:5);
    WRITELN('CH=', '', CH:3, '')
  END.
INPUT,
123
OUTPUT,
IO= 123
CH= ' '
```

FIG SMP04A PROGRAM

在执行这个程序时，它确实等待用户输入整数，然而，在输入整数123并回车后，它不等待用户输入字符，马上在屏幕上显示输入结果，程序运行结束。

显示结果表明，程序执行过程中自动输入了一个字符，它也是一个空格字符。

这种现象是怎样产生的呢？又如何去解决呢？

先看一下产生这个现象的原因。按规定，当程序执行到 READ 语句时，预先准备好内容在那里等待被读，而键盘输入恰恰办不到这一点，因为系统无法保证程序执行到 READ 语句时，用户一定打过键盘。为此，在OMSI PASCAL-1的编译程序中加了一点内容，即每当要从终端开始读入数据时，就首先人为地把一个空格字符做为被读来的头一个字符处理，这是加进来的，而不是真正读来的，这样虽然保证了原来对 READ 语句所做规定的有效性，但给读字符等却带来了新的问题，这是OMSI PASCAL-1语言未处理好的一个问题。这个问题可以用在读字符之前增加一个 READLN语句的办法来解决。

程序如FIG SMP03B PROGRAM所示。

```
PROGRAM SMP03B (INPUT, OUTPUT),
  VAR
    IO:INTEGER,
    CH:CHAR,
  BEGIN
    READLN,
    READLN(CH),
    IO:=ORD (CH);
    WRITELN ('CH=', '', CH, '');
  END.
```

```

        WRITELN ('IO=', IO:3)

    END.
INPUT,
#
OUTPUT,
CH=' #'
IO= 35
FIG SMP03B PROGRAM

```

对比它与 FIG SMP03A PROGRAM 后发现，在 READLN (CH) 语句之前加了一个 READLN 语句，程序运行时就会等待用户输入字符，在输入字符并后，屏幕显示的结果是正确的。

同理，将 FIG SMP04A PROGRAM 中 READ (CH) 之前的 READ (IO) 改成 READLN (IO)，形成新的程序，如 FIG SMP04B PROGRAM 所示。

```

        PROGRAM SMP04B (INPUT, OUTPUT);
    VAR
        IO: INTEGER;
        CH: CHAR;
    BEGIN
        READLN (IO);
        READLN (CH);
        WRITELN ('IO=', IO:5);
        WRITELN ('CH=', '''', CH:3, '''')
    END.
INPUT,
123
@
OUTPUT,
IO= 123
CH=' @'
FIG SMP04B PROGRAM

```

在运行这个程序时，它不仅等待用户输入整数，而且在输入整数123并回车后，它还等待输入字符。在输入字符@并回车后，终端显示程序运行结果，程序执行结束。

实践中出现的问题必须通过实践来解决，从解决的方法可以看出，空格字符的自动产生确实与换行 (LINE FEED) 有关。因此，在读字符的语句前面加一个语句 READLN，如果先读整数 I，再读字符，则用 READLN (I)，不用 READ (I) 就能自动消除空格字符，符合用户的实际需要。当然，增加一个读字符语句 READ (CH) 也是解决问题的另一种方法。然而，相比之下，还是 READLN 方法更简单，不会带来副作用。

事实上，空格字符问题在输入整数或实数时同样存在，只是由于在读一个整数或实数之前，都要首先把读来的空格字符“扔”掉，包括加进去的那个和从键盘打进去的那些（如果存在的话）。所以它不影响正常的输入。

例3-4 B 输入一个整数，计算其对应的字符，然后再求出该字符的序数值，最后输出这个字符及其序数值。

本例题通过输入大小不同的整数值，输出不同的结果，使读者熟悉和了解字符在计算机中的存贮方法，进一步掌握序数函数和字符函数之间的关系及有关要求。

程序如 FIG SMP04C PROGRAM所示。

```

PROGRAM SMP04C (INPUT, OUTPUT),
VAR
  I0, I1:INTEGER;
  CH:CHAR;
BEGIN
  READLN (I0);
  CH:=CHR(I0);
  I1:=ORD(CH);
  WRITE ('CH=', '''', CH, '''),
  Writeln(' I1=', I1:5)
END.

```

INPUT,	OUTPUT,
30	CH=' ' I1= 30
64	CH='@' I1= 64
80	CH='P' I1= 80
100	CH='d' I1= 100
-35	CH=']' I1= -35
-64	CH='@' I1= -64
-100	CH=' ' I1= -100
-400	CH='P' I1= 112
-10000	CH='P' I1= -16
500	CH='t' I1= -12

FIG SMP04C PROGRAM

在 PDP-11 计算机系统中，一个字分为两个字节，由 16 位二进制数码组成，其安排如下：



其中第 0—7 位表示低位字节，第 8—15 位表示高位字节。

一个整数占一个字（即二个字节），并用补码形式存放它的值。一个字符，本来只需占一个字节就够了，但为方便起见，也让它占二个字节。但此时，只有低位字节有用，即第 0—7 位有用。由于 ASCII 码字符的字符码只用 7 位二进制数码表示，所以只有第 0—6 位有用，第 7 位只在输入/输出过程中做奇偶校验用，而在内存中，往往把它清为零，同时也把高位字节清为零。

如何分析上面程序中的输入和输出之间的关系呢？可分四种情况来讨论。

1. 输入的整数小于 32，大于 0。如 30，这时找不到可以输出的字符，也不是空格字符，因为 CH=' ' 而不是 CH='␣'。但是，这个字符（实际上是不能输出的控制字符）的序数值，即字符码仍然存在，而且正好等于输入的整数值。这是因为字符本身无法存贮到计算机中，

它是通过字符码存贮在计算机中。因此，尽管控制字符不能输出，但是，其对应的字符码仍然存在， $I_1 = I_0$ 。

2. 输入的整数在32和126之间。如64和100。这种情况是一般的情况，字符可以输出，其对应的字符码就是输入的整数值， $I_1 = I_0$ 。

3. 输入的整数在-1和-128之间。如-64，-100和-35。这时的字符有的可以输出，如'@'和']'。有的不能输出，如"。但是，它们对应的字符码仍然是输入的整数值， $I_1 = I_0$ 。这是因为此时输入的整数值只用低八位就能表示了。

4. 输入的整数在(-128~127)之外。如-400，-10000以及500。这时为什么仍然输出字符？为什么其字符码 $I_1 \neq I_0$ ？

这个问题是第三个问题的发展，我们仍然看一看它们的补码表示。

-400的补码表示：1111111001110000末七位为112，字符为'p'序数值为+112；

-10000的补码表示：1101100011110000末七位为112，字符为'p'，序数值为-16；

500的补码表示：000000011110100末七位为116，字符为't'，序数值为-12；

思考题：

为什么同样输出字符'p'，末七位也相同，而序数值不一样呢？+112与-16之间有什么关系呢？（提示： $112 - (-16) = 128$ 。）

必须声明，通过终端键盘输入字符时出现的这种怪现象不具有普遍性，一般教科书及参考资料中都没有讨论到这个问题，因此，读者在实践过程中必须根据具体的情况（机器型号及语言版本）区别对待。至于消除空格符的其它办法，因为涉及到尚未学习过的内容，在这儿就从略了。

§8. 简单程序举例

在介绍了四种标准的数据类型、三种基本的运算和三个基本的语句的基础上，我们编几个简单的程序。

例3-5 输入一个整数和两个实数，然后整数乘100，两个实数相加，输出全部数据。程序如FIG SMP05 PROGRAM所示。

```
PROGRAM SMP05 (INPUT, OUTPUT);
VAR
  I1, I2: INTEGER;
  R1, R2, R3: REAL;
BEGIN
  READLN(I1, R1, R2);
  I2 := I1 * 100;
  R3 := R1 + R2;
  WRITELN ('I1=', I1:3);
  WRITELN ('I2=', I2:3);
  WRITELN ('R1=', R1:);
  WRITELN ('R2=', R2:6:3);
  WRITELN ('R3=', R3:6:3);
END.
```

```

INPUT,
12 67.89 56789.4321
OUTPUT,
I1= 12
I2=1200
R1= 6.789000E+01
R2= 56789.430
R3= 56857.320
INPUT,
-10 -2.5E10 500E-2
OUTPUT
I1=-10
I2=-1000
R1=-2.500000E+10
R2= 5.000
R3=-25000000000.000
      FIG SMP05 PROGRAM

```

这个程序运行的结果表明：

1. 输入实数时，既允许是十进位表示的，如 67.89，也允许是科学表示法表示的，如 $-2.5E10$ 。输入数据时的格式自由，允许不同于输出时的格式。如 $500E-2$ 。
2. 无论用户输入的实数有多少位有效数字，但是，它们在 PDP-11/03 系统中最多只有七位十进制有效数字，保证精确度为 6 位。比如，输入实数 56789.4321，共九位有效数字。但是，到机器中以及输出时的 R 为 56789.430。
3. 在输出实数时，既允许是科学表示法，按标准场宽输出，如 R1。也允许是十进位表示法，按用户自己定义的双场宽输出，如 R2 和 R3。其中场宽 1 允许突破，如 $R3 = -25000000000.000$ 。
4. 在突破场宽时，正整数的符号位会自动省略，如 $I2 = 1200$ 。但是，正实数的符号位照旧保存，以空格符表示，如 $R2 = _56789.430$ 。

例3-6A 输入整数 I，计算并输出 $J = (I-10)^2$ ， $Y = I\sqrt{I}$ 。请编制程序，实现上述功能。

程序如 FIG SMP06A PROGRAM 所示。

```

PROGRAM SMP06A (INPUT, OUTPUT),
VAR
  I, J :INTEGER,
  Y :REAL,
BEGIN
  READLN(I),
  J:=(I-10)*SQR(I-10),
  WRITELN('J=', J:6),
  Y:=I*SQRT(I),
  WRITELN('Y=', Y:8:2)
END.

```

INPUT,	OUTPUT,
20	J=1000
	Y=89.44
100	INTEGER ERROR
-16	J=-17576
	SQRT OF NEGATIVE
40000	BAD INTEGER
	INTEGER ERROR

FIG SMP06A PROGRAM

这个程序本身是比较简单的。在编译的过程中，它没有任何语法错误。但是，从输入和输出的关系中发现，如果输入的数据不合理，程序在执行时会产生错误。

在 $I=100$ 时，由于 $J=(100-10)^2=729000$ ，大于 32767，输出时显示“整数错误”(INTEGER ERROR)。

在 $I=40000$ 时，由于输入的数据本身超过整数范围，显示“整数有错”(BAD INTEGER)。由于计算出来的 $J>32767$ ，显示“整数错误”。

例3-6B，输入一个实数 X ，计算 $Y=x^2$ ，输出 Y 。编制程序实现上述功能。

程序如FIG SMP06B PROGRAM 所示。

```

PROGRAM SMP06B (INPUT, OUTPUT),
VAR
  X, Y:REAL,
BEGIN
  READLN(X),
  Y:=SQR(X),
  WRITELN('Y=', Y)
END.

```

INPUT,	OUTPUT,
-9.0E10	Y= 8.100000E+21
-9.0E0	Y= 8.100000E+01
9.0E00	Y= 8.100000E+01
9.0E-10	Y= 8.099999E-19
9.0E-20	Y= 8.100000E-39
-1.0E+40	TRAP
-9.0E+20	TRAP
9.0E-21	TRAP
1.0E-40	TRAP

FIG SMP06B PROGRAM

这个程序更简单。它没有任何语法错误。

如果输入的实数和计算的结果都在实数允许的范围内，程序运行时会输出结果。例如， X 为 $-9.0E10$ ， $9.0E-20$ 等。

当然，实数运算本身会有误差。比如， $X=9.0E-10$ ，按照通常方法计算， Y 应该为 $8.1E-19$ 。但是，实际上输出的 Y 为 $8.099999E-19$ 。

如果输入的实数或计算结果的绝对值太大，则产生“上溢”。如 $X=-1E40$ 是输入实数

的绝对值太大。

再如 $X = -9.0 \text{ E } 20$ 是计算出的 Y 的绝对值 ($8.1 \text{ E } 41$) 太大。

如果输入的实数或计算结果的绝对值太小, 则产生“下溢”。如 $X = 1 \text{ E } -40$ 是输入实数的绝对值太小。 $X = 9.0 \text{ E } -21$ 是计算出的 Y 的绝对值 ($8.1 \text{ E } -41$) 太小。

无论是“上溢”还是“下溢”, 程序运行时都会进入陷阱 (TRAP)。

必须指出, 尽管OMSI PASCAL-1 语言规定的实数范围为 $(-10^{+38} \dots -10^{-38})$ 和 $(10^{-38} \dots 10^{38})$ 。但是, 实践表明, 它允许超过一些。

例3-7. 输入两个整数。计算它们的和数, 整除后的商数。比较两个整数的大小。编制程序输入、计算、比较并输出结果。

程序如FIG SMP07 PROGRAM所示

```
PROGRAM SMP07 (INPUT, OUTPUT);
VAR
  A, B, S, D:INTEGER;
  L, E, G:BOOLEAN;
BEGIN
  READLN(A, B);
  S:=A+B;      D:=A DIV B;
  L:=A<B;      E:=A=B;
  G:=A>B;
  WRITELN('A=', A:5);
  WRITELN('B=', B:5);
  WRITELN('S=', S:5);
  WRITELN('D=', D:5);
  WRITELN('L IS', L);
  WRITELN('E IS', E);
  WRITELN('G IS', G)
END.
INPUT,
90 50
OUTPUT,
A=    90
B=    50
S=   140
D=     1
L IS FALSE
E IS FALSE
G IS  TRUE
INPUT,
-32767 1
OUTPUT,
A=-32767
B=      1
S=-32766
```

```

D = -32767
L IS TRUE
E IS FALSE
G IS FALSE
FIG SMP07 PROGRAM

```

例3-8 输入两个字符，分别求出它们的前导值、后续值和字符码，并且要求字符码以八进制输出。

程序如 FIG SMP08A PROGRAM 所示。

```

PROGRAM SMP08A (INPUT, OUTPUT),
VAR
  A1, A2, B1, B2, C1, C2:CHAR,
  D1, D2:INTEGER,
BEGIN
  READLN(A1, A2);
  B1:=PRED(A1);      B2:=PRED(A2);
  C1:=SUCC(A1);      C2:=SUCC(A2);
  D1:=ORD(A1);       D2:=ORD(A2);
  WRITELN('A1=', '', A1, '');
  WRITELN('B1=', '', B1, '');
  WRITELN('C1=', '', C1, '');
  WRITELN('D1=', D1:-5);
  WRITELN('A2=', '', A2, '');
  WRITELN('B2=', '', B2, '');
  WRITELN('C2=', '', C2, '');
  WRITELN('D2=', D2:-5)
END.
INPUT,
A~
OUTPUT,
A1=' '
B1=''
C1=':'
D1= 40B
A2='A'
B2='@'
C2='B'
D2= 101B
FIG SMP08A PROGRAM

```

这个程序运行的结果表明，自动加入的空格字符赋给了第一个变量A1，而且空格字符没有前导值；键盘输入的第一个字符A赋给了变量A2，而键盘输入的第二个字符~没有赋给任何变量，是一个多余的数据。输出的八进制数以字母B作为标志。

如果在读字符语句READLN (A1, A2) 之前加上一句 READLN，程序如FIG SMP08B 所示。

```

PROGRAM SMP08B (INPUT, OUTPUT),
VAR
  A1, A2, B1, B2, C1, C2:CHAR,
  D1, D2:INTEGER,
BEGIN
  READLN;
  READLN(A1, A2);
  B1:=PRED(A1);    B2:=PRED(A2);
  C1:=SUCC(A1);    C2:=SUCC(A2);
  D1:=ORD(A1);    D2:=ORD(A2);
  WRITELN('A1=', ' ', A1, ' ');
  WRITELN('B1=', ' ', B1, ' ');
  WRITELN('C1=', ' ', C1, ' ');
  WRITELN('D1=', D1:-5);
  WRITELN('A2=', ' ', A2, ' ');
  WRITELN('B2=', ' ', B2, ' ');
  WRITELN('C2=', ' ', C2, ' ');
  WRITELN('D2=', D2:-5);
END
INPUT,
A~
OUTPUT,
A1='A'
B1='@'
C1='B'
D1= 101B
A2='~'
B2='}'
C2=''
D2= 176B

```

FIG SMP08B PROGRAM

这个程序运行的结果表明，自动加入空格字符已经清除，键盘输入的两个字符A ~分别赋给变量A1和A2，而且字符~没有后续值。

必须指出，字符输入不同于整数和实数，两个字符之间不能用空格字符隔开，否则，那个空白字符就作为第二个字符起作用了。

思考题：

在FIG SMP08B PROGRAM中，如果输入字符A后就回车，会产生什么现象？为什么？

本章小结

本章的重点是通过三个基本语句进行简单的程序设计。计算圆柱体积的程序是沟通本章的主要桥梁。它既有常量定义，又有变量说明；它既有整数类型变量，又有实数类型变量；

它的整数型变量只有一个：H，而实数型变量却有两个：R和V。它既有赋值语句，又有读写语句，写语句中既有字符串，又有变量。

读语句中有三个问题要注意：

1. 输入数据的个数必须满足变量个数，输入数据的类型与变量类型要一致；
2. READ与READLN的区别；
3. 字符输入的特殊性及解决问题的办法。

写语句中的难点是场宽问题。尤其是自定义场宽问题。主要掌握这样的原则：场宽只影响输出的精度，不影响计算机运算；场宽与数据值发生矛盾时，以保证数据正确为主，兼顾场宽定义为辅。

本章习题

1. 下列表达式哪些是正确的？如能确定求出其表达式的值。哪些是非法的，错误在哪儿？表达式中，I为整型变量，R为实型变量，A、B、C、D为布尔型变量。

- (1) ODD (I) OR ODD (I+1) ;
- (2) ROUND (-65.3) < TRUNC (-65.3) AND A;
- (3) (A AND (B AND NOT B)) OR NOT (C OR (D OR NOT D))
- (4) ROUND (125/R) MOD 1;
- (5) NOT (A AND B) = NOT (NOT A AND NOT B);

2. 假定已作了如下常量定义和变量说明：

CONST

PI=3.14159; X='ABCD';

VAR

M, N: INTEGER; A, B: REAL;

P, Q: BOOLEAN; C, D: CHAR;

指出下列语句是否有效，并说明理由。

- (1) M:=TRUNC(B)*A+A;
- (2) P:=P+Q;
- (3) Q:=P AND (ORD (C1)≠CHR(M));
- (4) C:=X;
- (5) M:=N MOD A;
- (6) 'C':='D';
- (7) READ (M, A, P, C, X);
- (8) WRITE (N, B, Q, PI);

3. 下列常量定义和变量说明中，哪些是有效的，哪些是无效的？

CONSTANT

H=200; C='A'; D:=2;

PI=3.14159; B=0 OR 1 OR 2;

VAR

1A, 2B, 3C: CHARACTER;

A1, B2, C3: INTEGER;

B1, C2: BOOLEAN;

4. 用PASCAL语言的表达式表示下列各算式：

$$(1) - (x^2 + y^2) \cdot z^4; \quad (2) 3\sin^2(2x+7) + 5\cos(3x+4) + 7$$

$$(3) \cos((x^2 - 2xy + y^2) \cdot t + e^{-6});$$

$$(4) \frac{\frac{A+4}{8}}{B+4.5} - C;$$

5. 已知摄氏温度与华氏温度的关系式 $C = \frac{5}{9}(F - 32)$, 求华氏温度 F 为 30, 50, 70, 90, 100 度时的摄氏温度编一程序实现之。

6 绿化某大道, 大道两旁栽 4 排树, 每隔 5 米一棵, 每棵树苗 5 元, 大道全长二千米, 求这次绿化树苗费共需多少元? 编一程序实现之。

第四章 流程的控制

在第三章的程序中只用到读语句、赋值语句和写语句。程序执行部分的结构是顺序结构。就是说，所有语句都是按书写的顺序依次执行。

除此之外，当然还应有其它语句。

本章将讨论三类构造型语句。构造型语句一般由几个成份语句按照一定的规则组成。这些成份语句可以是基本语句，也可以是构造型语句。

构造型语句的组成规则，决定着程序执行部分的结构，它解决流程的控制问题。流程的控制有三种情况，一是顺序地执行，如复合语句；二是条件地执行，如条件语句；三是重复地执行，如重复语句。程序执行时的结构可以是顺序结构、分支结构或循环结构。

本章前四节主要讨论条件语句以及分支结构问题。后四节主要讨论重复语句以及循环结构问题。

§1 如 果 语 句

首先解一个最简单的方程。

例 4-1A 已知方程 $AX+B=0$ ，输入整数A和B，求一元一次方程的解X。

分析：

在一般情况下，对输入数据并不提出特殊的要求。因此，程序很简单。

程序如 FIG CTL01A PROGRAM 所示。

```
PROGRAM CTL01A (INPUT, OUTPUT);  
VAR  
  A, B:INTEGER;  
  X:REAL;  
BEGIN  
  READLN (A, B);  
  X:=-B/A  
END.  
INPUT,  
5      -1000  
FIG CTL01A PROGRAM
```

然而，在这个程序运行时，如果变量A输入的是零，则不仅方程无解，而且程序运行时一定出错。只有A不等于零时，方程才会有解，这就是“条件”。

“条件语句”中往往包含几个成份语句。它根据具体条件选择执行其中某一个成份语句。在PASCAL语言中，条件语句有两种：如果语句和情况语句。

如果语句规定，在一定条件下执行某一种成份语句，否则执行另一种成份语句，或者不执行任何成份语句。

如果语句有两种形式，其格式如图 4.1-1 所示。

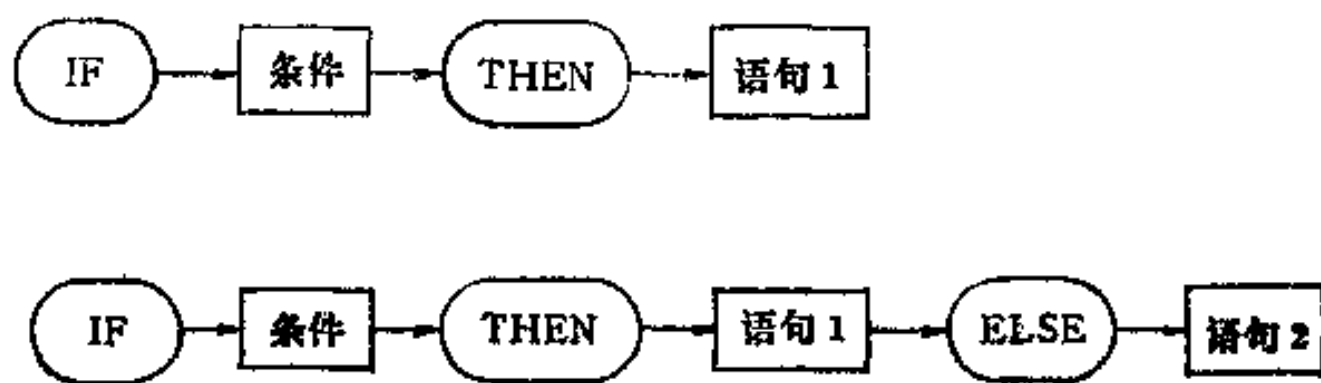


图 4.1-1 如果语句的格式

其中：

IF, THEN 和 ELSE 是如果语句的标志，都是保留关键字。

条件一般是布尔表达式，最简单的条件是一个布尔类型变量。

如果语句在执行时，其流程为分支结构。流程图如图4.1-2所示。

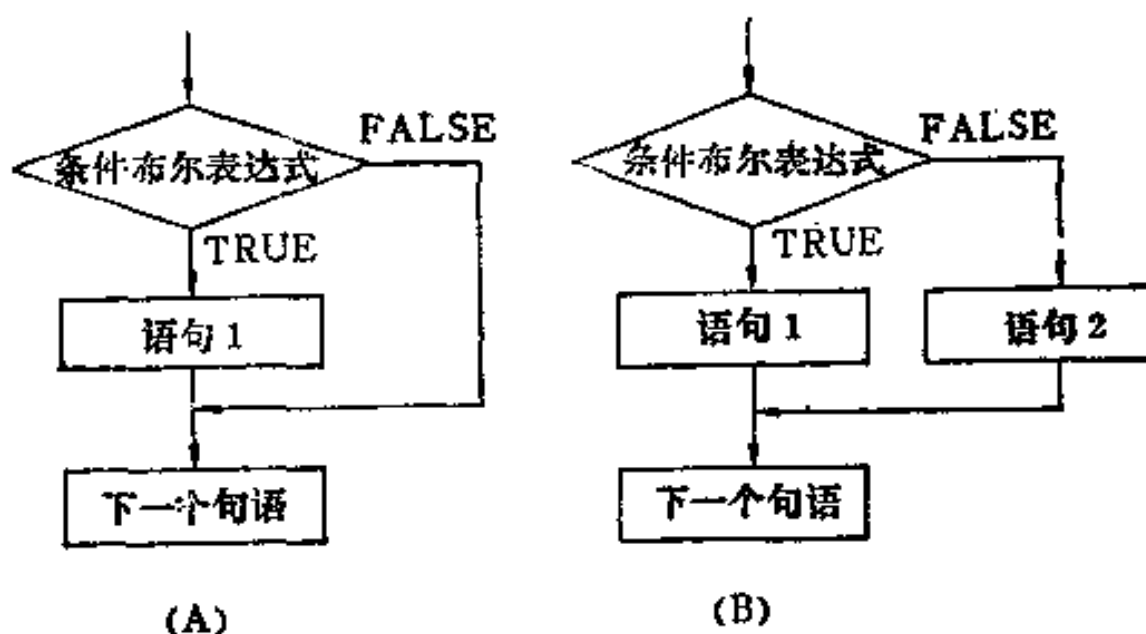


图 4.1-2 “如果语句”流程图

其中：

图 (A) 描述的是无 ELSE 部分的如果语句的流程图；

图 (B) 描述的是有 ELSE 部分的如果语句的流程图。

流程图是形象地描述程序设计思想的好方法。它比较直观。通常用一个框表示某一段程序（一般包含一个或几个语句），用一个箭头表示执行的路径及方向。

为了简化起见，本书中只用四种图样作为流程图的标志，如图4.1-3所示。

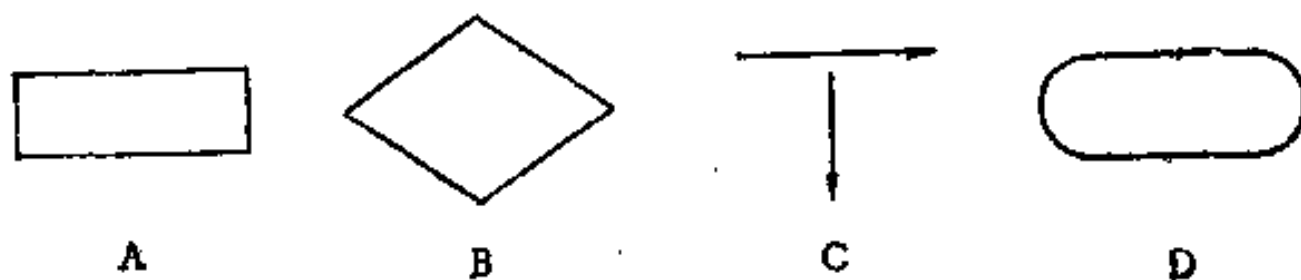


图 4.1-3 流程图的标志

A 为矩形框，又叫叙述框。它表示输入、计算及输出等各种功能；

B 为菱形框，又叫判断框。表示从几个可以选择的路径中，判断选取其中某一个路径。在条件语句及重复语句中有着广泛的应用；

C 为箭头。它表示流程的路径和方向；

D 为半圆框。它表示流程的开始和结束。

请注意，流程图的标志不同于语法图，关于语法图及其标志见附录一。

在编制程序时，尤其是编制复杂的程序时，框图能够帮助人们进行程序设计或了解程序结构，框图还能帮助人们尽快熟悉程序设计人员的思想，看懂程序。因此，充分发挥框图的优越性，是程序设计人员的职责。

如果语句的两种形式，其共同点在于都必须判断条件；在条件成立时，即布尔表达式计算结果为真时，都必须执行一个成份语句 1。

如果语句两种形式的主要区别在于：在条件不成立时，第一种形式不执行任何成份语句，直接转去执行下一个语句。而第二种形式必须先执行另一个成份语句 2，然后才转去执行下一个语句。

例如，对于例 4-1 A 的问题，编制程序时，必须考虑到条件：“只有 A 不等于零时，方程才会有解”。

新程序如 FIG CTL01B PROGRAM 所示。

```
PROGRAM CTL01B (INPUT, OUTPUT);
VAR
  A, B:INTEGER;
  X:REAL;
BEGIN
  READLN (A, B);
  IF      A<>0
    THEN  X:=-B/A
  END.
INPUT,
5      -1000
FIG CTL01B PROGRAM
```

这个程序在运行时，如果 $A = 0$ ，那么不进行任何运算，也不输出任何信息。

假如希望在 $A = 0$ 时输出一段信息：“输入数据错误！” (ERROR IN INPUT DATA₁)，那么，新程序如 FIG CTL01C PROGRAM 所示。

```
PROGRAM CTL01C (INPUT, OUTPUT);
VAR
  A, B:INTEGER;
  X:REAL;
BEGIN
  READLN (A, B);
  IF      A<>0
    THEN  X:=-B/A
    ELSE  WRITELN ('ERROR IN INPUT DATA1 ')
  END.
INPUT,
5      -1000
OUTPUT,
INPUT,
```

OUTPUT,

ERROR IN INPUT DATA;

FIG CTL01C PROGRAM

这个程序在运行时，如果输入的A不等于零，计算机就进行计算，但不显示结果。如果输入的A为零，则显示“ERROR IN INPUT DATA;”。

从语法上有一点必须强调，在语句1的后面、ELSE的前面没有分号；初学者常常在这儿犯错误。

思考题1：

图4.1-4所示两个“如果语句的功能是否相同？有无区别？

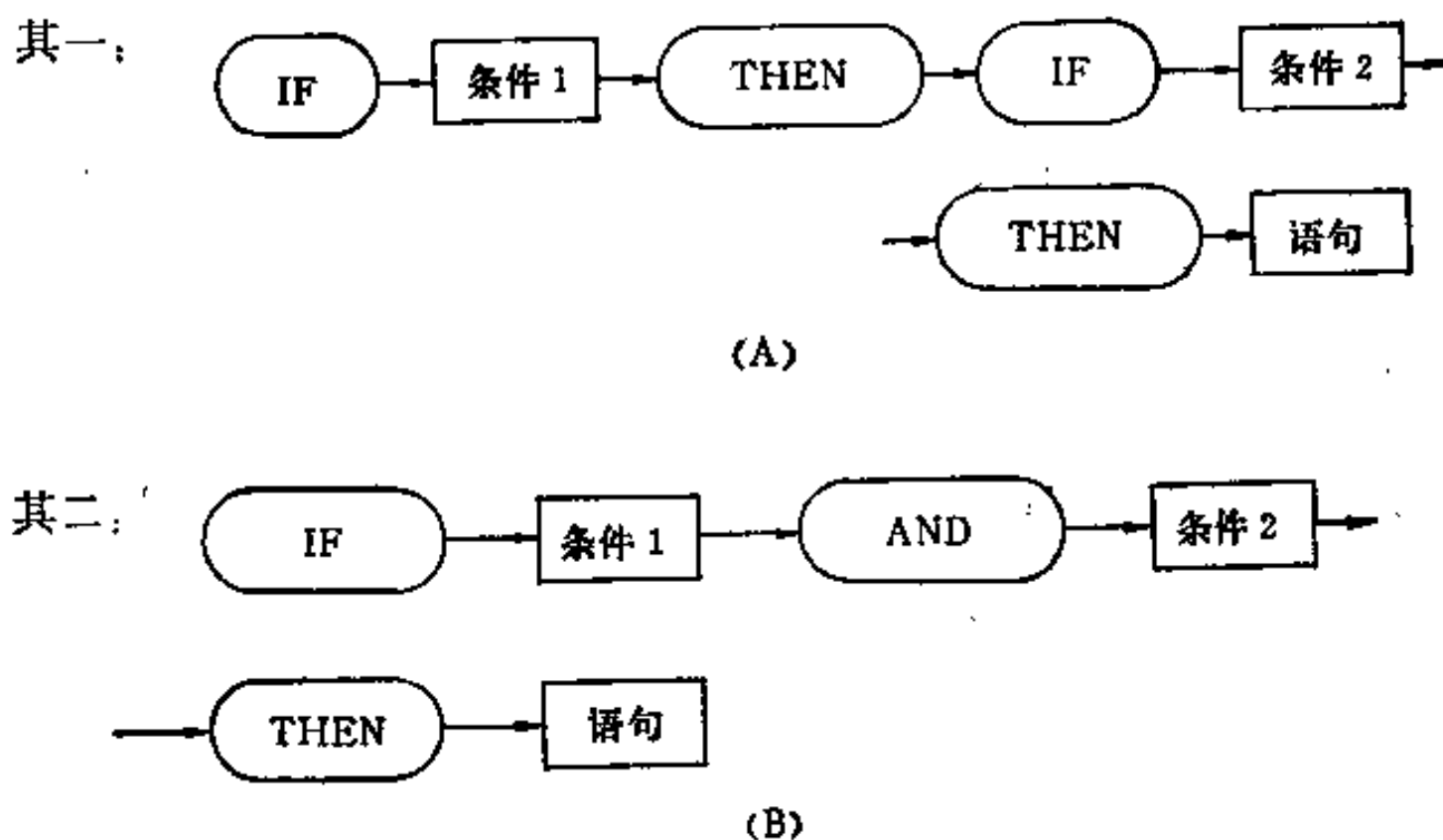


图 4.1-4 两种如果语句功能的比较

分析：

第一，两者的最终执行结果是相同的。

第二，前者适用于条件1复杂，不容易具备的情况下，在执行时马上跳过去，从而节省运行时间。

第三，后者适用于两个条件具备的可能性差不多的情况下，程序比较短，书写方便，但运行时间要长一些。

思考题2：已知如果语句：

IF $A > 0$

THEN IF $B < 0$

THEN $Y := 1$

ELSE $Y := 0;$

这里的ELSE是否定 $B < 0$ 还是否定 $A > 0$ 呢？

分析：

由于ELSE只对最靠近的IF起作用，因此这里的ELSE否定的是 $B < 0$ 的情况，

如果希望ELSE否定 $A > 0$ 的情况，则改成如下形式：


```

IF A > 0
  THEN BEGIN
    IF B < 0
      THEN Y := \
    END
  ELSE Y := 0,

```

这里的 ELSE 最靠近 IF $A > 0$ ，而与 IF $B < 0$ 毫无关系，因为有 BEGIN.....END 隔开了。BEGIN.....END 的问题将在下一节讨论。

§2 复 合 语 句

我们继续讨论一元一次方程的求解问题。

例 4-1B 已知一元一次方程 $AX+B=0$ ，输入 A 和 B，计算并输出 X 的值。

这个例题要求在 A 为零时显示“输入数据错误”。在 A 不为零时做两件事：计算并输出方程的解。这两件事作为一个整体来处理，用复合语句来实现。

新程序如 FIG CTL01D PROGRAM 所示。

```

PROGRAM CTL01D (INPUT, OUTPUT);
VAR
  A, B: INTEGER;
  X: REAL;
BEGIN
  READLN (A, B);
  IF A <> 0
  THEN
    BEGIN
      X := -B/A;
      WRITELN ('X=', X:7:2)
    END
  ELSE WRITELN ('ERROR IN INPUT DATA; ')
  END.
INPUT:
5    -1000
OUTPUT:
X = -200.00
INPUT:
0    1000
OUTPUT:
ERROR IN INPUT DATA;
FIG CTL01D PROGRAM

```

这个程序的运行结果中包含输出方程的解。其余与程序 FIG CTL01C PROGRAM 相同。

复合语句中，一最包含一系列成份语句，它按照书写的顺序执行其成份语句。

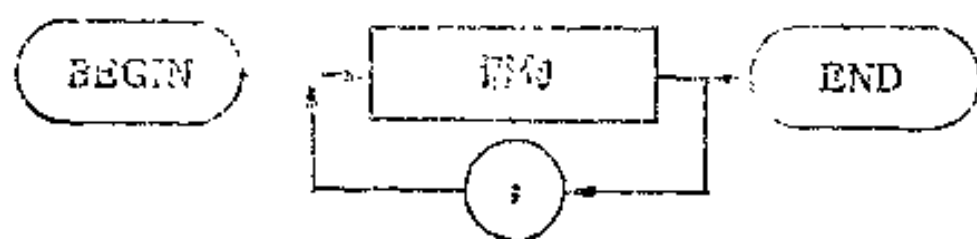


图 4.2-1 复合语句的格式

复合语句的格式如图4.2-1所示。

其中，BEGIN 和 END 是复合语句的标志，都是保留关键字。

复合语句中成份语句之间用语句隔离符分号隔开，复合语句与后续语句之间也用分号隔开。复合语句中的

BEGIN.....END 与程序执行部分的 BEGIN.....END 意义不同，前者表示的是一种语句，后者表示的是程序结构的一个部分。程序执行部分的 END 后面必须是句号，而不能是分号。

思考题：

在 FIG CTL01D PROGRAM 中，如果去掉程序执行部分内部的 BEGIN 和 END，会出现什么现象？能否正常运行？

必须指出，是否要把一个成份语句组织成复合语句，要看具体需要。比如上一节结束时的复合语句就非要不可，目的是为了确定语句的从属关系：

```
IF A > 0
  THEN BEGIN
    IF B < 0
      THEN Y := 1
    END
```

```
ELSE Y := 0 ;
```

当然，这个语句稍加修改，就可以不用复合语句了。例如：

```
IF A < = 0
  THEN Y := 0
  ELSE IF B < 0
    THEN Y := 1 ;
```

复合语句是程序中经常被应用的一种语句，在以后的学习中会发现这样的规律：在标号语句，当语句，循环语句，如果语句，情况语句和开域语句的保留关键字 DO, THEN, ELSE 以及标号后面必须跟随一个以上语句时，必须使用复合语句，也就是说用 BEGIN 和 END 把一个以上的语句包起来。

§3 情 况 语 句

在如果语句中，根据一个条件，会有两种可能。在实际生活中，情况要复杂得多。假如有许多条件可以选择，那么就必须根据“具体情况具体分析”的原则，用情况语句去解决。

情况语句中有许多成份语句。它根据条件的具体情况选择执行其中的某一个成份语句。它是如果语句的发展。

情况语句的标准格式如图4.3-1。

为了形象化地表达情况语句，可以用图4.3-2所示格式。

其中：

CASE, OF, END 是情况语句的标志，都是保留关键字；

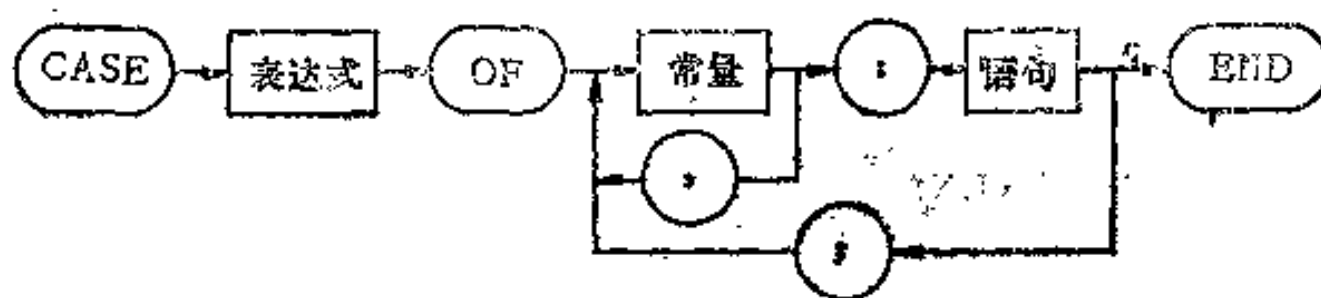


图 4.3-1 情况语句的标准格式

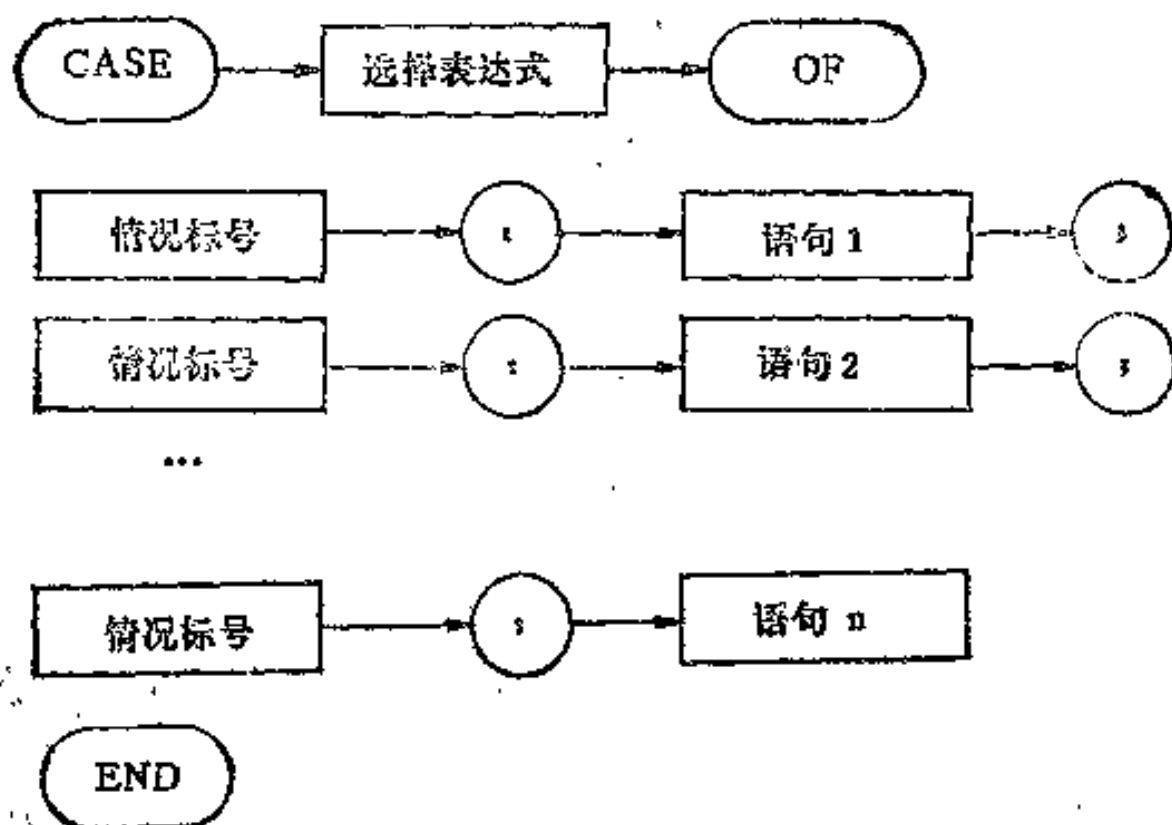


图 4.3-2 情况语句的另一种表示格式

选择表达式通常是一个变量，一般是整数型变量或字符型变量。布尔型变量也可以使用。实数型变量不能使用。用户定义的其它类型的变量必须根据具体情况决定。

情况标号一般是一个常量，如整数、字符。情况标号的数值是程序执行过程中产生的，因此，不必在程序的说明部分加以说明。情况标号在情况语句中的次序是任意的，不一定按照从小到大的次序排列，但是，数据类型应与表达式计算结果的数据类型相一致。

情况语句，允许几种情况执行一个成份语句（甚至是同一个成份语句），也允许一种情况执行许多语句，但是，这许多语句必须用 BEGIN 和 END 包起来，形成复合语句，作为统一的成份语句。如果什么情况都不是，则不执行任何成份语句，立即跳出情况语句。

例 4-2 输入一个整数，在终端上输出对应输入整数的用英文表示的星期几。其中星期日用整数 0 表示。

程序如 FIG CTL02A PROGRAM 表示：

```
PROGRAM CTL02A (INPUT, OUTPUT);
VAR
  WEEK : INTEGER;
BEGIN
  READLN (WEEK);
  CASE WEEK OF
    0 :
      WRITELN ('SUNDAY');
    1 :
      WRITELN ('MONDAY');
```

```

2:      WRITELN ('TUESDAY');
3:      WRITELN ('WEDNESDAY');
4:      WRITELN ('THURSDAY');
5:      WRITELN ('FRIDAY');
6:      WRITELN ('SATURDAY')
END
END.
INPUT:      OUTPUT:
2           TUESDAY
6           SATURDAY
0           SUNDAY

```

FIG CTL02A PROGRAM

在这个程序运行时，假如输入一个负数，或者输入一个大于 6 的整数，则程序不输出任何结果，也不指出逻辑错误。

OMSI PASCAL-1 语言对情况语句有扩充。它允许除情况标号外，再加上一项：

ELSE 语句 n+1

这一项表示所有情况标号之外做什么。

例如，在例 4-2 的程序中，加上一条：如果输入的整数在 0 至 6 之外，则显示输入数据错误。

新程序如 FIG CTL02B PROGRAM 所示。

```

PROGRAM CTL02B (INPUT, OUTPUT);
VAR
    WEEK :INTEGER;
BEGIN
    READLN (WEEK);
    CASE WEEK OF
0:      WRITELN ('SUNDAY');
1:      WRITELN ('MONDAY');
2:      WRITELN ('TUESDAY');
3:      WRITELN ('WEDNESDAY');
4:      WRITELN ('THURSDAY');
5:      WRITELN ('FRIDAY');

```

6:

```
WRITELN ('SATURDAY');
ELSE
WRITELN ('ERROR IN INPUT DATA; ')
END
```

END.

INPUT:

2

6

0

-5

8

OUTPUT:

TUESDAY

SATURDAY

SUNDAY

ERROR IN INPUT DATA;

ERROR IN INPUT DATA;

FIG CTL02B PROGRAM

在情况语句中包含 ELSE 功能时，必须注意以下四点：

1. ELSE 必须在所有情况标号之后，不允许插在中间；
2. ELSE 与它前面的那个语句之间，必须有语句隔离符——分号。这一点不同于 IF... THEN...ELSE 形式的如果语句。

如：CASE WEEK OF

```
6: WRITELN ('SATURDAY');
ELSE WRITELN ('ERROR IN INPUT DATA; ')
END;
```

3. ELSE 后面直接跟 语句 $n+1$ ，中间不允许用冒号隔开；这一点不同于情况标号。语句 $n+1$ 可以是简单语句或构造型语句；

4. 在语句 $n+1$ 与 END 之间不允许用分号，这一点不同于一般的情况语句。

在一个情况语句中，情况标号必须是互不相同的，不允许有二义性。也就是说，在一个情况语句中，任何一个情况标号只能出现一次。

例如，在编制日历的程序中，有一个情况语句如下：

```
CASE MONTH OF
1: WRITELN ('JAN');
2: WRITELN ('FEB');
:
12: WRITELN ('DEC');
2: DAY:=28;
4, 6, 9, 11: DAY:=30;
1, 3, 5, 7, 8, 10, 12: DAY:=31
END;
```

在编译这个程序时，指出这里的错误是：情况标号重复 (DUPLICATE CASE LABEL)。这是因为每个情况标号都使用了两次。

如果将这—个情况语句拆成两个情况语句，则没有任何语法错误。

```
CASE MONTH OF
1: WRITELN ('JAN');
```

```

2: WRITELN ('FEB');
:
:
12: WRITELN ('DEC')
END;
CASE MONTH OF
2 : DAY := 28;
4,6,9,11 : DAY := 30;
1,3,5,7,8,10,12 : DAY := 31;
END;

```

例 4-3 已知一元二次方程 $AX^2+BX+C=0$ ，输入 A、B、C 后，根据不同情况计算并输出方程的根。

分析：

根据 A、B、C 的输入数据，可以分成七种情况。分别如下：

0：A、B、C 都为零，输出“输入错误”；

1：A 和 B 为零，C 不为零，输出“方程无解”；

2：A 为零，B 不为零，输出方程的单根；

3：A 不为零，C 为零，输出方程的双根，其中一个根为零；

在 A 和 C 都不为零时，根据 $D=B^2-4\cdot A\cdot C$ 的结果可分成三种情况：

4： $D=0$ ，输出方程的两个相等实根；

5： $D>0$ ，输出方程的两个不等实根；

6： $D<0$ ，输出方程的两个不等复数根。

程序如 FIG CTL03A PROGRAM 所示。

```

PROGRAM CTL03A (INPUT, OUTPUT),
VAR
A, B, C, D, K: INTEGER;
RE, IM: REAL;
BEGIN
  (* PART 1 ..... INPUT DATA *)
  READLN (A, B, C)
  (* PART 2 ..... DEFINE CASE *)
  IF (A=0) AND (B=0) AND (C=0)
    THEN K:=0;
  IF (A=0) AND (B=0) AND (C<>0)
    THEN K:=1;
  IF (A=0) AND (B<>0)
    THEN K:=2;
  IF (A<>0) AND (C=0)
    THEN K:=3;
  IF (A<>0) AND (C<>0)
    THEN
      BEGIN

```



```

    D:=SQR (B) - 4 * A * C;
    RE:=-B/ (2 * A);
    IM:=SQRT (ABS (D))/ (2 * A);
    IF D=0
    THEN K:=4;
    IF D>0
    THEN K:=5;
    IF D<0
    THEN K:=6;
    END;
    (*PART 3 ..... CASE STATEMENT*)
CASE K OF
0: WRITELN ('ERROR IN INPUT');
1: WRITELN ('UNSOLVABLE');
2: WRITELN ('ROOT IS ', -C/B:6:2);
3: WRITELN ('ROOTS ARE ', -B/A:6:2, ' AND 0.0');
4: WRITELN ('BOTH ROOTS ARE ', RE:6:2);
5:
    BEGIN
        WRITELN ('ROOTS ARE', (RE+IM):6:2);
        WRITELN ('      AND', (RE-IM):6:2);
    END;
6:
    BEGIN
        WRITELN ('ROOTS ARE COMPLEX');
        WRITELN ('      ',RE:6:2, '+I*', IM:4:2);
        WRITELN (' AND ',RE:6:2, '-I*', IM:4:2);
    END
END
END.

```

INPUT.			OUTPUT:
0	0	0	ERROR IN INPUT;
0	0	7	UNSOLVABLE;
0	10	2	ROOT IS -0.20
2	3	0	ROOTS ARE -1.50 AND 0.0
2	0	0	ROOTS ARE -0.00 AND 0.0
1	2	1	BOTH ROOTS ARE -1.00
1	5	6	ROOTS ARE -2.00 AND -3.00
1	0	-16	ROOTS ARE 4.00 AND -4.00
1	1	1	ROOTS ARE COMPLEX; -0.50 + I*0.87 AND -0.50 - I*0.87

1 0 16

ROOTS ARE COMPLEX:

-0.00 + 1*4.00

AND --0.00 - 1*4.00

FIG CTLO3A PROGRAM

为了便于比较, 我们用“如果语句”编制了一个程序, 如 FIG CTLO3B PROGRAM 所示. 同样用来解一元二次方程 $AX^2 + BX + C = 0$.

```
PROGRAM CTLO3B (INPUT, OUTPUT),
VAR
  A, B, C, D :INTEGER;
  RE, IM :REAL;
BEGIN
  READLN (A, B, C);
  IF A=0
  THEN
    IF B=0
    THEN
      IF C=0
      THEN WRITELN ('ERROR IN INPUT,')
      ELSE WRITELN ('UNSOLVABLE,')
      ELSE WRITELN ('ROOT IS', -C/B:6:2)
    ELSE
      IF C=0
      THEN WRITELN ('ROOTS ARE ', -B/A:6:2, ' AND 0.0')
      ELSE
        BEGIN
          D:=SQR (B) -4*A*C,
          RE:=-B/ (2*A),
          IM:=SQRT (ABS (D))/ (2*A),
          IF D=0
          THEN WRITE ('BOTH ROOTS ARE', RE:6:2)
          ELSE
            IF D>0
            THEN
              BEGIN
                WRITELN ('ROOTS ARE', (RE+IM):6:2),
                WRITELN (' AND', (RE-IM):6:2)
              END
            ELSE
              BEGIN
                WRITELN ('ROOTS ARE COMPLEX,')
                WRITELN (' ', RE:6:2, '+1*', IM:4:2),
                WRITELN (' AND ', RE:6:2, '-1*', IM:4:2)
              END
            END
        END
      END
    END
  END
```

END	
END.	
INPUT:	OUTPUT:
0 0 0	ERROR IN INPUT;
0 0 7	UNSOLVABLE;
0 10 2	ROOT IS -0.20
2 3 0	ROOTS ARE -1.50 AND 0.0
2 0 0	ROOTS ARE -0.00 AND 0.0
1 2 1	BOTH ROOTS ARE -1.00
1 5 6	ROOTS ARE -2.00
	AND -3.00
1 0 -16	ROOTS ARE 4.00
	AND -4.00
1 1 1	ROOTS ARE COMPLEX:
	-0.50 + I*0.87
	AND -0.50 - I*0.87
1 0 16	ROOTS ARE COMPLEX:
	-0.00 + I*4.00
	AND -0.00 - I*4.00

FIG CTLO3B PROGRAM

不难看出，用“情况语句”要好一些，它格式整齐，简单清晰，便于分析和阅读。

必须指出，在确定情况标号时，各个情况标号的条件必须绝对地独立。不允许这种条件既满足情况标号M，又满足情况标号N。

思考题：

在 FIG CTLO3A PROGRAM 中，将下列语句

IF (A=0) AND (B=0) AND (C=0) THEN K:=0;

改成 IF A=0 THEN K:=0;

这时会发生什么现象呢？

§4 标号说明和转移语句

在各种算法语言中，几乎都有转移语句。然而，PASCAL 语言为了程序动态结构和静态结构的一致，不提倡使用转移语句。甚至国外有些版本的 PASCAL 语言不讨论转移语句。

考虑到用户的实际需要和编写程序的方便，本书仍然介绍转移语句。转移语句的功能是改变程序执行的顺序，跳过程序的某一部分转去执行另一部分，或者返回已经执行过的某一个语句使之重复执行。

通常，转移语句和条件语句配合使用，它作为条件语句的成分语句。因为无条件转移语句，它能使一部分程序永远不执行，这通常用在向前转移的程序中。这是使用转移语句的目的之一。至于出现死循环的问题，在个别特殊情况是有用的，但多数情况是程序设计错误所致：重复执行部分的条件永远不能改变，或者只在几个转移语句之间无限制循环。出现这种

死循环后，只能通过人工干预或由操作系统强迫程序停止运行。请注意，程序设计中一定要尽量避免这类错误。

在 PASCAL 语言中，转移语句的一般格式如图4.4-1；

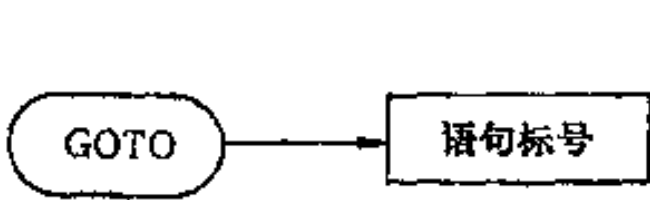


图 4.4-1 转移语句的格式

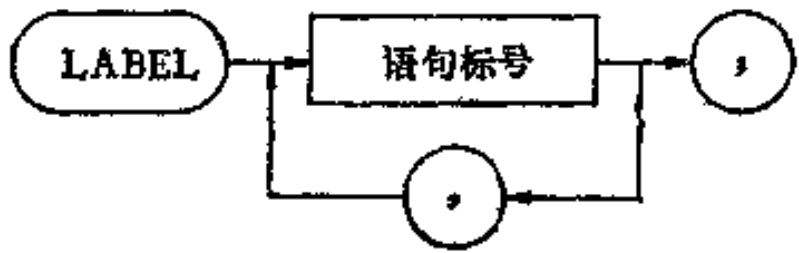


图 4.4-2 标号说明的格式

其中：

GOTO 是转移语句的标志，它是一个保留关键字。请注意，它不允许象其它某些算法语言一样写成 GO TO。

语句标号表示程序转移去执行的那个语句的标号。

在 PASCAL 语言中，凡是程序执行部分使用的语句标号，必须在程序说明部分进行标号说明。这一点不同于某些算法语言。标号说明的格式如图4.4-2所示。

其中：

LABEL 是标号说明的标志，它是一个保留关键字。

语句标号为整数，允许从 1 到9999。既可以几个语句标号共用一个 LABEL，也允许每个语句标号单独用一个 LABEL。语句标号一般是由小到大排列，可以是不连续的排列顺序。

在程序执行部分，语句标号后面必须有语句跟随，表示转移到这儿干什么。标号语句的格式如图4.4-3所示。



图 4.4-3 标号语句的格式

其中：

语句标号必须在程序说明部分已经说明；

语句可以是任何语句，基本语句和构造型语句都可以，甚至常常用空语句。但是使用什么样的语句必须慎重。请看下面的例题。

例4-4 输入一个实数，计算并输出其平方根。在实数小于零时显示“不能解”。

分析：

显而易见，这个问题不用转移语句也很容易解决，只要用条件语句就可以了。下面是条件语句和转移语句的结合应用。

程序如 FIG CTL04A PROGRAM 所示。

```
PROGRAM CTL04A (INPUT, OUTPUT),
LABEL
  10;
VAR
  X, Y:REAL;
BEGIN
  READLN (X);
  IF X<0
  THEN
    BEGIN
```

```

        WRITELN ('CANNOT SOLVE1 ');
        GOTO 10
    END;
    Y:=SQRT(X);
    WRITELN('Y=',Y:6:2);
10;
    END
INPUT,
9
OUTPUT,
Y= 3.00
INPUT,
-9
OUTPUT,
CANNOT SOLVE1

```

FIG CTL04A PROGRAM

在这个程序中，在X小于零时，执行的是复合语句。其一是写语句，其二是转移语句。而最后的标号语句中的语句是空语句。

在执行这个程序时，输入实数9时，输出其平方根为3.00；输入实数-9时，输出错误标志“不能解”(CANNOT SOLVE₁)。显然，这个程序是完全正确的。

如果将程序简单修改一下，在X小于零时，仅仅执行转移语句。而最后的标号语句中的语句是写语句。这往往是初学程序设计的同志习惯使用的方法。

程序如FIG CTL04B PROGRAM所示。

```

        PROGRAM CTL04 B (INPUT, OUTPUT),
        LABEL
            10;
        VAR
            X, Y:REAL;
        BEGIN
            READLN(X);
            IF X<0
            THEN GOTO 10 ;
            Y:=SQRT (X);
            WRITELN ('Y=', Y:6:2);
10;
            WRITELN ('CANNOT SOLVE1')
        END.
INPUT,
9
OUTPUT,
Y= 3.00
CANNOT SOLVE1
INPUT,

```

OUTPUT,

CANNOT SOLVE,

FIG CTL04B PROGRAM

在这个程序中，由于用户的疏忽，在输入 $X = 9$ 时，程序不仅正确地输出平方根，而且错误的指出“不能解”。这种矛盾的结论显然是由于程序的错误引起的。

下面看一个死循环的例子。

PROGRAM CTL04C (OUTPUT);

LABEL

1;

VAR

I:INTEGER,

BEGIN

I:=1,

1,

BEGIN

I:=I-1;

WRITELN ('I:=I-1;')

END;

IF I<100

THEN GOTO 1;

WRITELN ('THE VALUE OF I PASS OVER 100')

END.

FIG CTL04C PROGRAM

这是个很简单的程序，但它里面有死循环。因为 I 的初值小于 100，以后又用的是把 I 值减 1 操作。所以 IF 语句中的条件 $I < 100$ 总是成立的，就总是执行 GOTO 1，则在标号为 1 的语句与它下面的 IF 语句之间无休止的循环。这是因为标号为 1 的语句中有错误，应将减号改为加号，即 $I := I + 1$ 才对。这只是一个最简单的例子，很容易找出它里面的错误来。当程序比较大，转移比较多而判断是否要转移的条件比较复杂，影响这些条件的因素又比较多时，要查出这类错误就比较费事，应该在程序设计过程中谨慎对待。

条件语句、转移语句和标号语句的结合是分支结构程序常用的一种方法。由于转移语句不利于程序动态结构和静态结构的统一，稍不留心，则会产生某些错误。因此，程序设计时必须认真对待，仔细分析。

在情况语句中用到情况标号，在转移语句中用到语句标号。两者有什么区别呢？请看下面的程序。

PROGRAM CONTROL (INPUT, OUTPUT);

LABEL

10, 15;

VAR

K:INTEGER,

BEGIN

语句 1;

```

10: 语句2;
CASE K OF
  5: GOTO 15;
  10, 20: 语句3;
  15: 语句4;
END;
语句5;
15: 语句6;
语句7

```

END.

在这个程序中，七个语句可以为任何语句——基本语句或构造型语句。语句1、语句5和语句7前面没有任何标号，按一般规则执行。语句2和语句6之前的标号是语句标号，必须在程序说明中加以说明。语句3和语句4之前的标号是情况标号，不必在程序说明部分加以说明，它们由程序执行时选择。转移语句只能转移到语句标号处，不能转移到情况标号处。GOTO 15是转去执行语句6，不是转去执行语句4。几个情况标号（如10，20）共用一个语句是允许的，但是不允许几个语句标号共用一个语句。

转移语句在循环结构以及过程、函数中的应用有许多重要的规则，本书将陆续讨论。

§5 当语句

物质三态（气态、液态、固态）的转变是有条件的。例如，当水温降到摄氏零度以下时，水变成固体——冰；当水温升到摄氏100度以上时，水变成气体——水蒸汽。在PASCAL语言中，描述这类问题时，用当语句。

当语句规定：当条件成立时，重复执行成份语句。重复的次数取决于成份语句的执行情况。



图 4.5-1 当语句的格式

当语句的一般格式如图4.5-1；

其中：

WHILE 和 DO 是当语句的标志，都是保留关键字；

条件为布尔表达式；

语句为成份语句。成份语句个数超过一个时，必须构成复合语句。

当语句的流程图如图4.5-2所示。

例 4-5 程序员通过终端键盘输入字符，计算机进行计数。当输入的字符为‘?’之后，计算机停止计数。最后在终端显示器上输出计数的结果。

分析：

本题主要按照下述四个步骤进行——

1. 输入字符；
2. 判断字符：字符为‘?’时转步骤4；

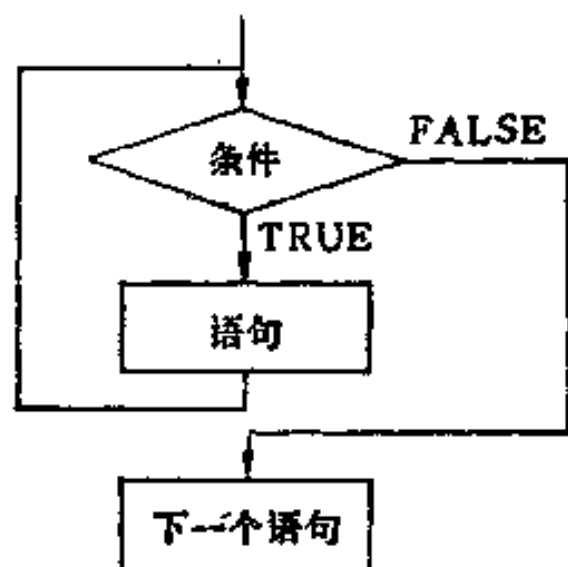


图 4.5-2 当语句流程图

3. 进行计数;
4. 输出计数结果。

程序如 FIG CTL05 PROGRAM 所示。

```

PROGRAM CTL05 (INPUT, OUTPUT),
CONST
  P='?';
VAR
  I:INTEGER;
  C:CHAR;
BEGIN
  I:=0;      READLN;
  READ(C);
  WHILE C<>P DO
    BEGIN
      I:=I+1;
      READ (C)
    END;
  WRITELN ('I=', I:2)
END.
INPUT;      INPUT;      INPUT;
ABCDE?      7654?321      ?ABCD
OUTPUT;     OUTPUT;     OUTPUT;
I=5          I=4          I=0

```

FIG CTL05 PROGRAM

根据这个程序和输入输出的情况回答下列问题:

1. 为什么要用 READLN 语句? 去掉它, 计数的结果会变化吗? 为什么?
2. 如果不断输入的字符为 AB# !CDEF 会输出计数结果吗? 为什么?
3. 如果将语句 READ (C) 都改成 READLN (C), 仍然输入上述各组字符, 都能正确显示计数结果吗? 为什么?
4. 如果将当语句中的 BEGIN 和 END 去掉, 仍然输入上述各组字符, 能否正确显示计数结果? 为什么?

从当语句的流程图可以看出, 当语句的主要特点如下:

1. 首先判断条件, 决定是否要执行成分语句。
2. 条件成立时, 继续执行成分语句。在正常使用的当语句中, 成分语句的执行结果应该能改变这一条件, 否则会出现死循环。这是程序设计中容易出现的严重错误, 应当尽力避免之。条件不成立时, 则执行下一个语句。
3. 如果条件总是成立, 则不断重复执行成份语句。比如 WHILE I=1 DO 语句; 这在特殊情况下是可以使用的。这是一个死循环的特例, 在这种程序运行时, 只能通过人工干预的方法或由操作系统强迫其停止运行。
4. 如果条件从一开始就不成立, 则不执行成分语句。比如 WHILE I<>1 DO 语句; 这种语句实际上是没有使用价值的。

下面讨论两个死循环的例子。例子很简单，但能说明一点问题。

```
PROGRAM CTL051;
VAR
  I: INTEGER;
BEGIN
  I:=0;
  WHILE I<100 DO
  BEGIN
    I:=I*2;
    WRITELN ('I:=I*2;')
  END;
  WRITELN ('THE VALUE OF I PASS OVER 100')
END.
```

这个程序与 CTL04C 有类似的错误。WHILE 语句中的条件 $I < 100$ 总成立，它的成份语句不会使这一条件改变。因为 I 的初值为零， $I := I * 2$ 总是使 I 保持为零。则 WHILE 语句的执行是不会结束的。

```
PROGRAM CTL052;
VAR
  I, J: INTEGER;
BEGIN
  I:=0;
  WHILE I<100 DO
  BEGIN
    J:=I+2;
    WRITELN ('J:=I+2;')
  END;
  WRITELN ('THE VALUE OF I PASS OVER 100')
END.
```

这个程序的错误更明显，WHILE 语句的成份语句的执行结果与它的条件无关，即向 J 赋值并不会影响 $I < 100$ 这个循环条件。若把条件 $I < 100$ 改为 $J < 100$ 或把成份语句 $J := I + 2$ 改为 $I := I + 2$ ，那末就不再会是死循环了。

§6 直到语句

在日常生活中，经常遇到要重复办某一件事，直到条件改变为止。在 PASCAL 语言中，描述这类问题时，用直到语句。

直到语句规定：程序重复执行某些成份语句，直到条件成立时为止。重复的次数一般与成份语句的执行情况有关。

直到语句的一般格式如图 4.6-1。

其中：

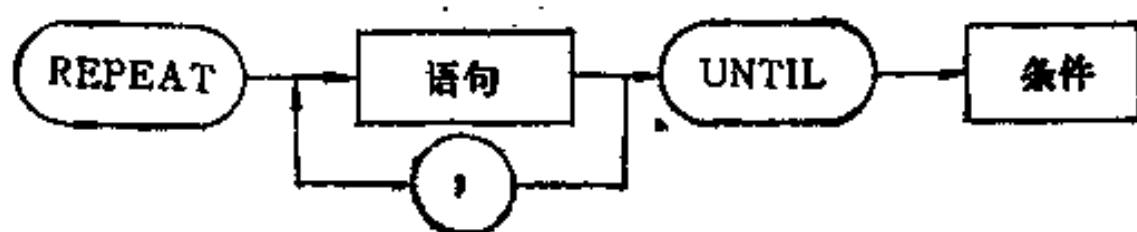


图 4.6-1 直到语句的格式

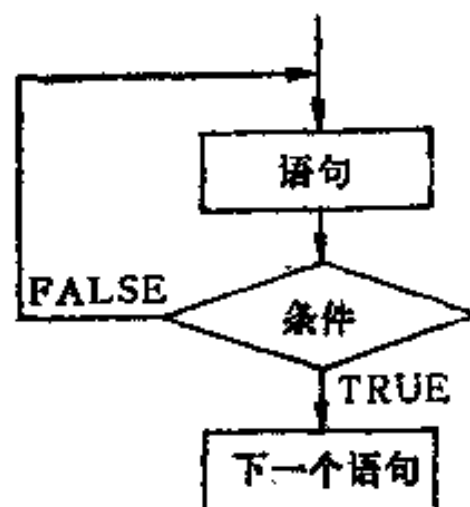


图 4.6-2 直到语句流程图

REPEAT 和 UNTIL 是直到语句的标志，都是保留关键字；

条件一般为布尔表达式；

语句为成份语句。成份语句超过一个时，不必构成复合语句。

直到语句的流程图如图4.6-2所示。

例4-6程序员通过终端键盘输入六个整数，计算并输出它们的和数及平均数。

分析：

本题可以用多种方法解决，下面是用直到语句的大致步骤。

1. 输入一个整数；
2. 进行求和；
3. 进行计数；
4. 判断条件：不满六个数时转步骤 1；
5. 求平均数；
6. 输出和数及平均数。

程序如 FIG CTL06A PROGRAM 所示。

```

PROGRAM CTL06A (INPUT, OUTPUT),
CONST
  H=6;
VAR
  I, N, SUM:INTEGER,
  AVERAGE:REAL;
BEGIN
  N:=0; SUM:=0;
  REPEAT
    READ (I);
    SUM:=SUM+I;
    N:=N+1
  UNTIL N=H;
  AVERAGE:=SUM/H;
  WRITELN ('SUM IS', SUM:5);
  WRITELN ('AVERAGE IS', AVERAGE:5:2)
END.
INPUT,
  
```

1 2 3 4 5 6

OUTPUT,

SUM IS 21

AVERAGE IS 3.50

FIG CTL06A PROGRAM

思考题:

1. 如果程序员输入五个整数后回车, 程序会输出计算结果吗? 为什么?
2. 如果程序员输入七个整数后回车, 程序会输出计算结果吗? 为什么?
3. 如果将语句 READ (I) 改成 READLN (I), 程序员输入六个整数后回车, 程序会输出计算结果吗? 为什么?

通过这些思考题进一步回顾一下读语句执行的规律以及 READLN 与 READ 之间的区别。

从直到语句的流程图可以看出, 它的主要特点如下:

1. 不管情况如何, 首先执行成分语句, 然后判断条件。
2. 条件不成立时, 继续执行成分语句; 执行成份语句的结果, 应能改变这一条件, 否则就会出现死循环。这与当语句中的情况类似, 应尽量避免之。条件成立时, 执行下一个语句。

3. 如果条件总是不成立, 则构成死循环, 就会不断重复执行成份语句。比如:

REPEAT [语句]

UNTIL I<>I;

这是出现的死循环中最简单的情况, 也可以说它是个特例。这在特殊情况下是可以使用的。为了退出这一死循环, 只能通过人工干预的办法或由操作系统迫使其停止运行。

4. 如果条件从一开始就成立, 则它也要执行一次成份语句。比如,

REPEAT [语句]

UNTIL I=I;

对于这一点, 请读者在使用时要特别注意。

这里不再给出死循环的例子, 它与当语句中的情况非常类似。

思考题:

对照当语句和直到语句的四条主要特点, 找出它们的共同点及不同点。

§7 循 环 语 句

经济发展计划的年增长率, 银行存款和国际贷款的利息, 这些比例关系随着年月的推移, 总是重复地计算。另外会经常遇到有规则的统计, 次数确定的循环计算, 读入或输出一定量的数据文件等。在PASCAL语言中, 推述这类问题时, 常用循环语句。

循环语句规定, 当控制变量在给定的范围内变化时, 它重复执行成分语句。重复的次数是由语句中的控制变量的初值与终值决定的, 一般与成份语句的执行情况无关。

循环语句一般有两种格式, 如图4.7—1所示。

其中:

FOR和DO是循环语句的标志, 都是保留关键字。

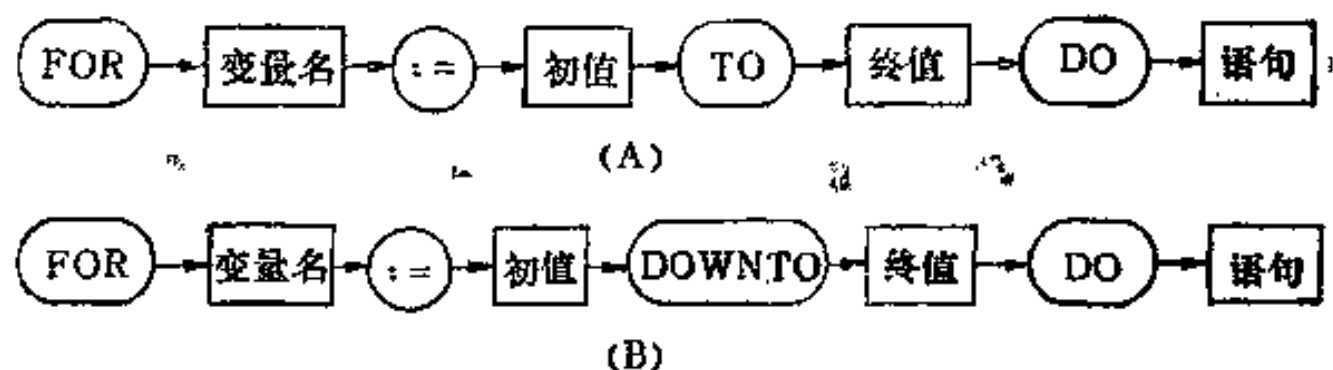


图4.7—1 循环语句的两种格式

变量名作为控制变量使用，它必须属于有序的数据类型。它常用整数类型和字符类型变量。也可以用枚举类型和子界类型变量。但不允许为实数类型变量。

初值和终值一般为算术表达式。它们的运算结果必须与控制变量属于同一种类型。语句为成分语句。成分语句超过一个时，必须构成复合语句。成份语句组成循环体。

$:=$ 是赋值运算符。

TO 是控制变量递增型循环语句的标志，它要求终值大于初值；DOWNTO 是控制变量递减型循环语句的标志，它要求终值小于初值。它们都是保留关键字。

在PASCAL语言中，循环语句没有步长这一项。如果控制变量属于整数类型，那么递增型的步长为+1，递减型的步长为-1。对于其他类型的控制变量，其步长也有类似的规定。

下面以整数类型控制变量为例，画出下列两个语句的流程图：

FOR $I:=I_1$ TO I_2 DO 语句；

FOR $I:=I_1$ DOWNTO I_2 DO 语句；

流程图如图4.7—2所示。

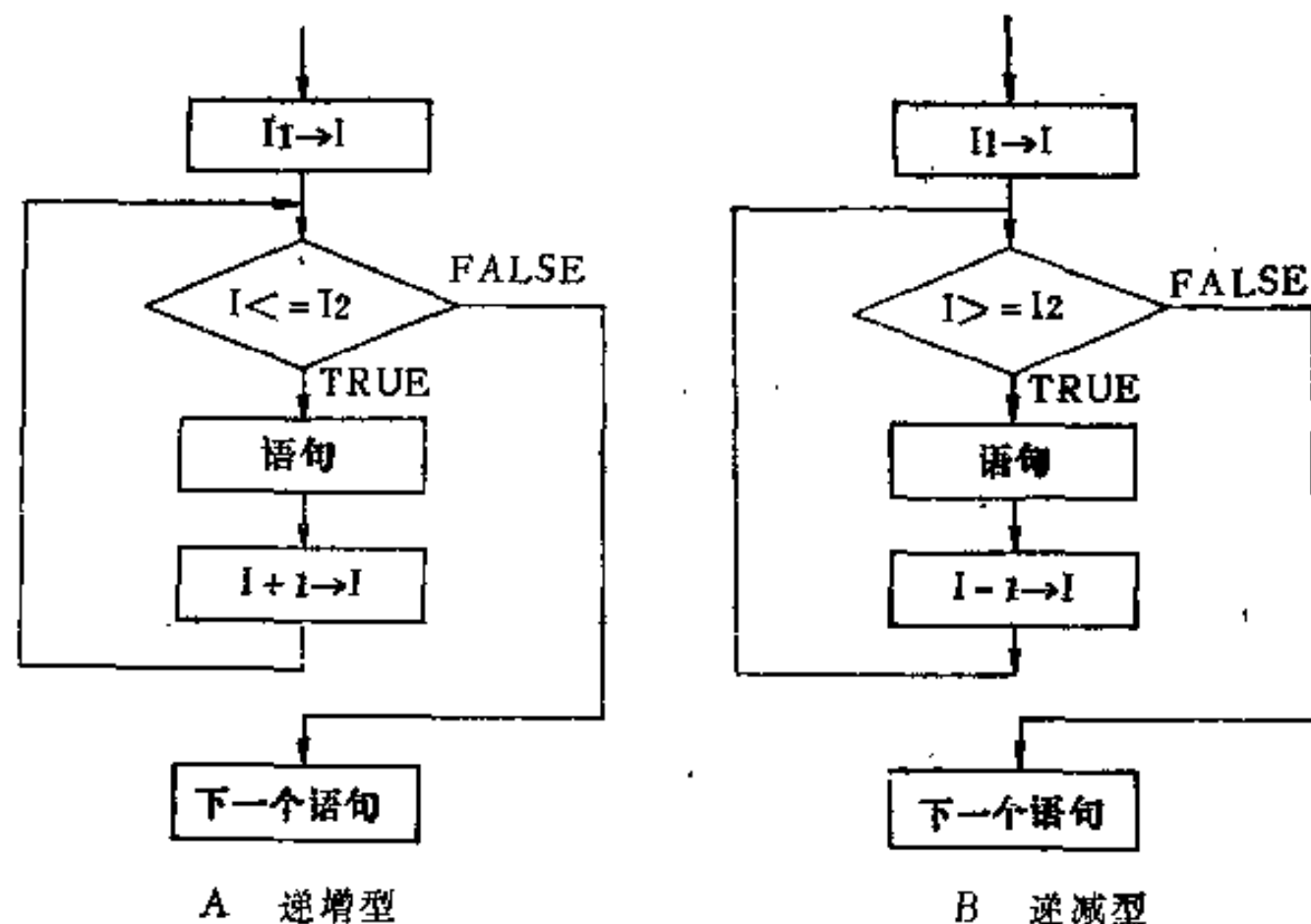


图4.7—2 循环语句流程图

图A是递增型循环语句流程图， $I+1 \rightarrow I$ 的工作是由机器自动完成的。因此，不需要在程序中书写赋值语句 $I:=I+1$ 。否则，步长就变为2。

图B是递减型循环语句流程图。只要修改控制条件为 $I \geq I_2$ ，同时将 $I+1 \rightarrow I$ 改为 $I-1 \rightarrow I$ 。其余与递增型循环语句相同。

从上面的递增型流程图中，可以看出循环语句的执行过程如下：

1. 将初值赋给控制变量;
2. 判断循环条件是否成立。如果条件不成立, 则转步骤 6;
3. 条件成立时执行循环体;
4. 修改控制变量;
5. 返回步骤 2;
6. 结束循环过程, 执行下一个语句。

必须指出, PASCAL 语言的循环语句不同于某些其它算法语言的循环语句。它是先判断循环条件是否成立, 后执行循环体。这一点与当语句非常类似。

对于不同的循环条件, 循环语句的执行情况也不相同。对于递增型循环语句:

1. 如果 $I_2 < I_1$, 则循环条件不成立。程序运行时不执行循环体, 立即转去执行下一个语句。
2. 如果 $I_2 = I_1$, 则循环条件成立。程序运行时只执行一次循环体。
3. 如果 $I_2 > I_1$, 则循环条件成立。程序运行时重复执行循环体, 重复次数为 $I_2 - I_1 + 1$ 。
4. 由于判断条件是 $I \leq I_2$, 不是 $I = I_2$, 因此, 退出循环时 $I = I_2 + 1$ 而不是 $I = I_2$ 。

例4—7某工厂1980年的产值为100万元, 计划年增长率为5%。请计算并输出1985年, 1990年, 1995年和2000年的产值。

分析:

本题主要解决两个问题:

1. 按5%的年增长率计算每年的产值: 即下一年产值是本年产值的1.05倍。
2. 每五年输出一次产值。

采用循环语句计算产值, 共循环20次。

采用条件语句输出产值。

程序如FIG CTL07 PROGRAM所示。

```

      PROGPAM CTL07(OUTPUT);
VAR
      Y :INTEGER;
      V, R:REAL;
BEGIN
      V:=100.0;      R:=0.05;
      Y:=1980;
      WRITELN('1980 YEAR: 100.00');
      FOR Y:=1981 TO 2000 DO
      BEGIN
          V:=V*(1+R);
          IF Y MOD 5=0
          THEN WRITELN(Y:4, ' YEAR ', V:8:2)
      END
      END.
OUTPUT:
1980  YEAR   : 100.00
1985  YEAR   : 127.63

```

```

1990  YEAR  : 162.89
1995  YEAR  : 207.89
2000  YEAR  : 265.33
      FIG CTL07  PROGRAM

```

其中:

Y代表年(YEAR);

V代表产值(VALUE);

R代表年增长率(RATE);

循环体为一个复合语句。

思考题:

在这个程序执行完之后, Y的值是多少? 为什么?

在循环体中, 引用控制变量是正常的, 它不会影响流程的控制。例如:

```
FOR I:=1 TO 5 DO Y:=SQR(I);
```

其中 I 既是控制变量, 又是平方函数的自变量。这是常有的现象。

然而, 在一般情况下, 不应该对控制变量进行修改, 大多数 PASCAL 语言版本禁止程序员在循环体内修改控制变量, OMSI PASCAL—1 版本虽然不做此种检查, 但建议大家尽量不要这样使用, 因为任意改变步长容易出错。例如:

```

FOR I:=1 TO 5 DO
  BEGIN
    Y :=SQR (I);
    I := 5 * Y
  END ;

```

在这个程序中, 由于修改了控制变量 I, 循环一次后, I 变为 6。(想一想, 为什么不是 5), 就退出了循环。

在特殊情况下, 为了程序精练, 提高运算速度等原因, 既然 OMSI PASCAL—1 语言不做此种检查, 我们也可以去改变步长以满足某种需要。但这是一种不好的程序设计习惯。

例4—8 编制程序输出 $1^2, 4^2, 7^2, \dots$ 自变量最大为20。

分析:

这里的1, 4, 7是不连续的。递增值为3, 不是1。因此, 可以采用步长为3的方法。

在循环体内, 完成三件工作——

1. 计算平方值;
2. 输出平方值;
3. 改变循环语句的步长。

程序如FIG CTL08 PROGRAM所示。

```

      PROGRAM CTL08 (OUTPUT);
      VAR
        I, Y:INTEGER;
      BEGIN
        : WRITELN('I':5, 'Y':10);
        FOR I:=1 TO 20 DO
          BEGIN

```



```

        Y:=SQR(I);
        WRITELN(I:5, Y:10);
        I:=I+2
    END
END.

```

OUTPUT:

I	Y
1	1
4	16
7	49
10	100
13	169
16	256
19	361

FIG CTL08 PROGRAM

其实，只要将循环体部分改成如下形式：

```

IF (I - 1) MOD 3 = 0
THEN
BEGIN
    Y := SQR (I) ;
    WRITELN (I : 5 , Y : 10)
END;

```

程序运行的结果是相同的，程序的结构更清楚些。

或者改用如下程序：

```

PROGRAM CTL08A (OUTPUT) ;
VAR
    I , Y , K : INTEGER ;
BEGIN
    WRITELN ('K' : 5 , 'Y' : 10) ;
    K := 1 ;
    FOR I := 1 TO 7 DO
        BEGIN
            Y := SQR (K) ;
            WRITELN (K : 5 , Y : 10);
            K := K + 3
        END
    END.

```

这里 I 只控制计算与输出的次数，而用 K 做为求平方函数的自变量，程序的结构更清楚。

思考题：

1. 改变步长时，为什么用赋值语句 $I := I + 2$ ，不用 $I := I + 3$ ？

2. 程序执行完之后, 1 的值是多少? 是20, 21还是22? 为什么?

3. 例4—7中每五年输出一次产值能否改成修改步长的方法? 为什么?

当语句、直到语句和循环语句的共同点之一是能重复执行成分语句。因此, 有些问题可以用其中任何一种语句来解决。几种重复语句的不同点也是很多的, 必须具体情况具体分析。

例如, 输入六个整数, 计算并输出其和数及平均数。

这不仅可以用直到语句来解决, 也可以用当语句和循环语句来实现, 如FIG CTL06B PROGRAM和FIG CTL06C PROGRAM所示

```
PROGRAM CTL06B (INPUT, OUTPUT),
CONST
  H=6;
VAR
  I, N, SUM:INTEGER;
  AVERAGE:REAL;
BEGIN
  WHILE N<H DO
    BEGIN
      READ (I);
      SUM:=SUM+I;
      N:=N+1
    END;
  AVERAGE:=SUM/H;
  WRITELN ('    SUM IS', SUM:5);
  WRITELN ('AVERAGE IS', AVERAGE:5:2)
END.
INPUT:
1 2 3 4 5 6
OUTPUT:
    SUM IS    21
AVERAGE IS  3.50
    FIG CTL06B    PROGRAM
```

```
PROGRAM CTL06C (INPUT, OUTPUT);
CONST
  H=6;
VAR
  I, N, SUM:INTEGER;
  AVERAGE:REAL;
BEGIN
  FOR N:=1 TO H DO
    BEGIN
      READ (I);
      SUM:=SUM+I
    END;
```

```

    AVERAGE:=SUM/H;
    WRITELN ('    SUM IS ', SUM:5);
    WRITELN ('AVERAGE IS ', AVERAGE:5:2)
END.
INPUT:
1 2 3 4 5 6
OUTPUT:
    SUM IS    21
    AVERAGE IS 3.50
    FIG CTL06C  PROGRAM

```

一般地说，如果重复的次数与其成份语句执行情况无关时，通常用循环语句比较好。否则用当语句或直到语句，它们重复的次数取决于成份语句执行情况。

学过其它算法语言的读者，常常对PASCAL语言的循环语句没有改变步长的功能不习惯。事实上，PASCAL语言的直到语句与当语句已经圆满地解决了这个问题。例如，步长为0.15时，计算0到1之间各有关实数的平方，可以用下列语句：

```

REPEAT
    Y:=SQR(X);
    WRITELN ('X=', X:5:2, ' Y=', Y:7:4);
    X:=X+0.15
UNTIL X>1.0;

```

§8 多重循环语句

上一节的循环语句中，循环体虽然包含几个成份语句，但是成份语句中没有循环语句。这是最简单的循环。

如果在循环体中包含一个循环语句，则称为二重循环。在第二层循环中又包含一个小的循环，则称为三重循环。……，继续一层套一层（嵌套），则形成多重循环。

例4—9 编制程序输出一张“九——九”乘法口诀表。要求格式如下：

```

*   1   2   3   4   5   6   7   8   9
1   1
2   2   4
:
:
8   8  16  24  32  40  48  56  64
9   9  18  27  36  45  54  63  72  81

```

分析：

这是一个二重循环问题。一方面横向参数要变化，另一方面纵向参数也要变。

本题将横向参数变化作为外循环，纵向参数变化作为内循环。内循环次数受外循环的约束。

流程图如图4.8—1所示。

程序如FIG CTL09 PROGRAM所示。

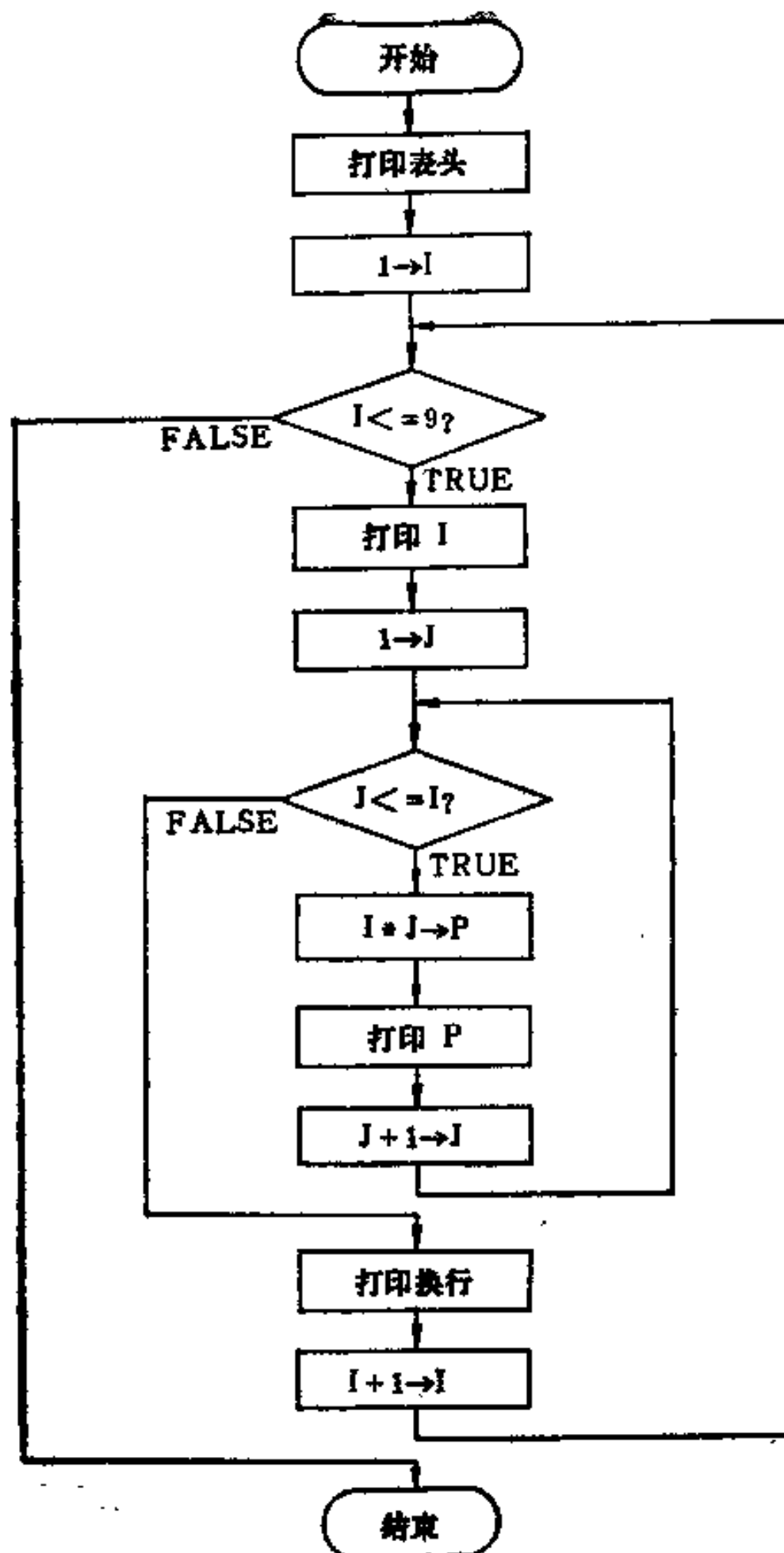


图4.8—1 “九九乘法口诀表”流程图

```

PROGRAM CTL09(OUTPUT);
VAR
  I, J, P:INTEGER;
BEGIN
  WRITE(' * ':4, ' ':4);
  FOR I:=1 TO 9 DO WRITE(I:4);
  FOR I:=1 TO 2 DO WRITELN;
  FOR I:=1 TO 9 DO
    BEGIN
      WRITE(I:4, ' ':4);
      FOR J:=1 TO I DO
        BEGIN

```

```

        P:=I * J;
        WRITE(P:4)
    END;
    FOR J:=1 TO 2 DO WRITELN
END
END.
OUTPUT:

```

•	1	2	3	4	5	6	7	8	9
1	1								
2	2	4							
3	3	6	9						
4	4	8	12	16					
5	5	10	15	20	25				
6	6	12	18	24	30	36			
7	7	14	21	28	35	42	49		
8	8	16	24	32	40	48	56	64	
9	9	18	27	36	45	54	63	72	81

FIG CTL09 PROGRAM

在这个程序中，大致分为两部分：

第一部分为前三条语句，它是一重循环语句和写语句的结合，用以输出乘法口诀表的表头。

第二部分为第四条语句开始到结束前的程序段。它是程序的核心部分，采用的是二重循环语句。

其中：

外循环的控制变量为 I，循环语句为

```
FOR I:=1 TO 9 DO BEGIN.....END
```

在BEGIN和END之间为外循环体。

在外循环体内又包含内循环。内循环的控制变量为 J，循环语句为

```
FOR J:=1 TO I DO BEGIN.....END;
```

在BEGIN和END之间为内循环体。内循环体解决计算和输出乘积的问题。

对照程序框图及程序，可以看出二重循环的执行过程如下：

1. 对外循环控制变量 I 赋初值；
2. 判断外循环控制条件：条件成立时执行外循环体，否则转向步骤 7；
3. 当执行到内循环时，对内循环控制变量 J 赋初值；
4. 判断内循环控制条件：条件成立时执行内循环体，否则转向步骤 6；
5. 执行完一次内循环体后，J 按步长增值，并返回步骤 2；
6. 继续执行外循环体。在执行完一次后，I 按步长增值，并返回步骤 2；
7. 退出循环，执行下一个语句。

关于多重循环的嵌套问题，跟其它算法语言一样，PASCAL 语言也有一些规定。用户在程序设计时，必须遵循如下规定：

1. 外循环体一般包括内循环及其它语句，但是内循环必须完全在外循环体内，内外循

环不得相互跨跨。

例如，将本题中二重循环部分改成如下形式：

```
FOR I:=1 TO 9 DO
  BEGIN
    WRITE(I:4);
    FOR J:=1 TO I DO P:=I * J
  END;
  WRITE(P:4);
```

这样做，虽然在语法上没有错误，但是程序执行的结果是不符合要求的。原因在于语句 `WRITE(P:4)` 应该在内循环体内，不应该放在内循环体外，更不应该放到外循环体外。

2. 内循环在外循环体中的位置是任意的，但是在分析流程时要避免造成混乱。

3. 内外循环的控制变量不能同名。例如：

```
FOR I:=1 TO 9 DO
  FOR I:=3 TO 7 DO 语句;
```

这种多重循环语句在语法上是错误的。

4. 外循环体允许包括几个内循环，这些并列的内循环允许用相同的控制变量。例如：

```
FOR I:=1 TO 9 DO
  BEGIN
    FOR J:=10 TO 20 DO 语句 1;
    FOR K:='A' TO 'Z' DO 语句 2;
    FOR J:=100 TO 200 DO 语句 3;
  END;
```

这种多重循环语句在语法上是没有错误的，完全能够正常运行。

本节讨论的多重循环的思想，不仅适用于循环语句，同样适用于当语句和直到语句。

循环语句中的成份语句也可以是当语句或直到语句，同样体现多重循环的思想。

当语句或直到语句中的成份语句可以是当语句、直到语句或循环语句中任一种语句，体现多重循环的思想。

§9 退出循环语句

在循环语句执行的过程中，如果满足了一定的条件，不希望继续循环，则可以立即退出循环以节省运行时间。

标准的PASCAL语言没有专用的退出循环语句。在一般情况下，可用在条件语句的成份语句中使用转移语句的办法来解决问题。例如

```
FOR I:=1 TO 100 DO
  BEGIN
    V:=(I+R) * V;
    IF V>130
      THEN GOTO 10
  END;
```

10. 语句

在这一段程序中，只要条件成立，就马上退出循环，执行下一个语句。但是，由于转移语句和标号语句执行的特点，编制程序时必须谨慎地考虑。

OMSI PASCAL—1 语言定义了一个专用的退出循环语句：EXIT；

在一般情况下，它总是作为条件语句的成份语句，很少单独使用。

例4—10 某工厂1980年的产值为100万元，计划最近几年的年增长率为6%。在产值超过200万元之后，计划年增长率降为4%。输入1980至2000年中任何一年的数值，计算并显示该年的产值。

分析：

因为本题有两个不同的年增长率，所以必须用两个相互独立又相互联系的循环语句来实现。

当产值小于200万元之前，年增长率为6%时，执行第一个循环语句。

当产值达到200万元之后，年增长率为4%时，退出第一个循环语句，转去执行第二个循环语句。

两个循环语句的共同特点是终值相同。

两个循环语句之间的相互关系是后者的初值为前者的控制变量 I 加 1，即 $I + 1$ 。

框图如图4.9—1所示。

Y 代表年份 (YEAR)；

R 代表年增长率 (RATE)；

V 代表产值 (VALUE)；

I 为第一个循环语句的控制变量；

J 为第二个循环语句的控制变量。

```
PROGRAM CTL10(INPUT,OUTPUT),
LABEL
  100;
VAR
  I, J, Y:INTEGER,
  V, R:REAL;
BEGIN
  READLN(Y);
  IF (Y<1980) OR (Y>2000)
  THEN
    BEGIN
      WRITELN('ERROR IN INPUT DATA 1'),
      GOTO 100
    END;
  V:=100.0; R:=0.06;
  FOR I:=1981 TO Y DO
    BEGIN
      V:=V*(1+R);
      IF V>200
```

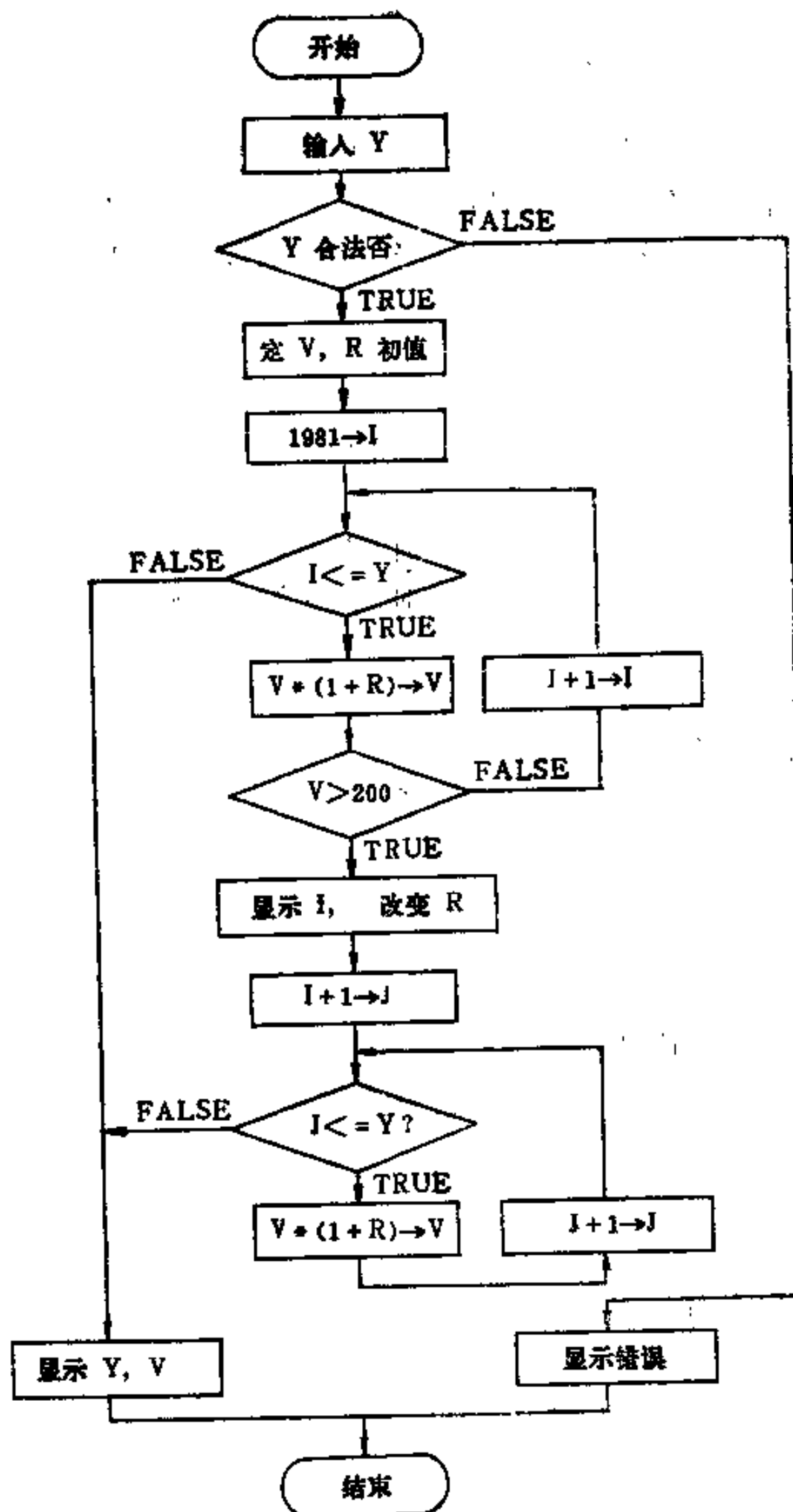



图4.9—1 例4—10的流程图

```

THEN
  BEGIN
    WRITELN('V>200 FROM', I:5, ' YEAR', ' ');
    R := 0.04;
    EXIT
  END
END,

```

```

    FOR J:=(I+1) TO Y DO V:=V*(1+R);
    WRITELN(Y:4,' YEAR:',V:8:2);
100:
    END.
INPUT:          OUTPUT:
1949            ERROR IN INPUT DATA;
1980            1980 YEAR: 100.00
1985            1985 YEAR: 133.82
1990            1990 YEAR: 179.08
1992            V>200 FROM 1992 YEAR;
                1992 YEAR:201.22
1995            V>200 FROM 1992 YEAR;
                1995 YEAR:226.34
2000            V>200 FROM 1992 YEAR;
                2000 YEAR:275.38
2049            ERROR IN INPUT DATA;

```

FIG CTL10 PROGRAM

在这个程序运行时，有以下几种不同的输入情况，并会有不同的执行结果。

1. 如果输入1980年到2000年中的任何一年，则总会显示那一年的产值数。
2. 如果输入的那一年产值不足200万元，则不会执行 EXIT 语句，也不会执行第二个循环语句的循环体。比如1985年。
3. 如果输入的那一年产值是首次达到200万元，则会执行 EXIT 语句，但不会执行第二个循环语句的循环体。比如1992年。
4. 如果输入的那一年的前一年或前几年的产值就已超过200万元，则不仅会执行 EXIT 语句，而且会执行第二个循环语句的循环体。比如，1995年。
5. 如果输入的那一年小于1980年或者大于2000年，则仅会输出“输入数据错误”(ERROR IN INPUT DATA₁)。

对于 EXIT 语句的使用，要注意以下几点：

1. EXIT 语句不仅能用在“循环语句”中，同样能用在“当语句”和“直到语句”中。而且仅能用在这三种重复语句中。
2. EXIT 语句执行时，只能退出循环。或广义地讲，不再重复。一般情况下并不退出整个程序，初学者常常误解。
3. 在多重循环的情况下，EXIT 语句退出的是直接控制它的那一层循环，并不退出整个循环。

例如，下面的程序退出第二层循环（控制变量为 J）

```

FOR I:=1 TO 10 DO
  BEGIN
    FOR J:=10 TO 20 DO
      BEGIN
        FOR K:=1 TO 15 DO Y:=I*J*K;
        IF Y>2000 THEN EXIT;

```

```

        WRITELN (I:5, J:5, K:5)
    END;
    WRITELN ('Y=', Y:5)
END;

```

转移语句和三种重复语句结合时，尤其是在多重循环或并列循环中被应用时，有许多重要的限制。主要有以下几点：

1. 在同一层内允许自由转移；
2. 允许从循环体内转向循环体外 反之不行；
3. 允许从内循环转向外循环，反之不行；
4. 不允许在并列的循环之间互相转移。

例如，一个程序的执行部分如下：

```

BEGIN
    IF A THEN GOTO 10;
    WHILE B DO
        BEGIN
            Y:=X;
            IF Y=0 THEN GOTO 20
        END;
10: FOR I:=1 TO 10 DO
        BEGIN
            Z:=5+10*I;
            IF Z>37 THEN GOTO 30
        END;
20: REPEAT
        P:=R>S;
        R:=P+1;
        IF P THEN GOTO 30
    UNTIL R=100;
30:
END.

```

这段程序没有语法错误。因为 GOTO 10 是在同一层中转移；GOTO 20 是从循环语句中转移出来；GOTO 30 是从直到语句中转移出来。它们都体现了由内而外的原则。标号语句 30，是两个 GOTO 30 的共同归宿，同样是允许的。

又如，一个程序的执行部分如下：

```

BEGIN
    GOTO 10;
    IF A THEN Y:=X
        ELSE 10: Z:=X;
    WHILE B DO
        BEGIN

```

```

        Y:=X;
    20:  Z:=X
    END;
GOTO 30;
FOR I:=1 TO 10 DO
    BEGIN
        Z:=5+10*I;
        40:  WRITELN(Z)
    END;
GOTO 40;
REPEAT
    P:=R<S;
    R:=R+1;
    30:  WRITELN(P)
UNTIL R=100;
GOTO 20
END.

```

这段程序有许多语法错误。因为所有标号语句都在构造型语句（条件语句和重复语句）之中。所有转移语句都在主程序中。这样的转移语句都是由外向内的转移，所以都是不允许的。

§10 应用举例

通过第三章和第四章的讨论，不仅解决了程序说明部分常用的标号说明、常量定义和变量说明的问题，而且将 PASCAL 语言常用的语句基本介绍完毕。

这一节，利用这些语句及基本概念，结合实际，综合解决一些问题。

例4-11 程序员通过终端键盘输入字符，其中 A, B, C 和 D 必须单独计数，其余字符统一计数，但遇到字符 '?' 号后不再计数。请编程序实现对输入字符进行分别计数并输出计数结果等功能。

分析：

由于输入的是字符，所以程序设计时必须考虑字符输入的特点；

由于输入字符的个数并不确定，因此，不能用循环语句。以采用直到语句为好；

由于需要分别计数，可以使用情况语句，特别是用带 ELSE 功能的情况语句；

程序如 FIG CTL11 PROGRAM 所示。

```

PROGRAM CTL11(INPUT, OUTPUT);
VAR
    IA, IB, IC, ID, IO: INTEGER;
    CH: CHAR;
BEGIN
    IA:=0; IB:=0; IC:=0;

```

```

        ID:=0,   IO:=0,
        READLN,
        REPEAT
            READ(CH),
            CASE CH OF
                'A' : IA:=IA+1,
                'B' : IB:=IB+1,
                'C' : IC:=IC+1,
                'D' : ID:=ID+1,
                ELSE IO:=IO+1
            END
        UNTIL CH='?',
        Writeln('    A:', IA:5),
        Writeln('    B:', IB:5),
        Writeln('    C:', IC:5),
        Writeln('    D:', ID:5),
        Writeln('OTHER:', IO:5)
    END.
INPUT:
ABCDEFGH1234567890BCCDD#:<>XYZD? AAP
OUTPUT:
    A:      1
    B:      2
    C:      3
    D:      4
OTHER:    22
        FIG CTL11 PROGRAM

```

思考题:

1. 输入的最后四个字符? AAP参加计数没有? 为什么?
2. 如果将 READ(CH)改成 READLN(CH), 照样输入这些字符, 会出现什么现象?

例4-12 输入正整数 N , 计算 $S_1 = \frac{1}{N+1}$. 根据数学的知识, 数列 $a_k = \frac{1}{K(K+1)}$ 的前 N

项和: $S_2 = \frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \frac{1}{3 \times 4} + \cdots + \frac{1}{N(N+1)}$. 计算并输出 S_1 和 S_2 , 看一看两者是否相等。

分析:

1. 输入的 N 必须为整数, 否则指出“输入数据错误”;
2. 计算数列时, 因为 $K(K+1)$ 容易超过整数的范围, 因此, 在用赋值语句时, 不用语句 $A:=1/(K * (K+1))$, 而用语句 $A:=1/(K+1)/K$.
3. 计算前 N 项和时, 必须用循环语句, 通过不断累加数列 a_k 来实现。
4. 按照数学归纳法, S_1 应该与 S_2 相等。

程序如 FIG CTL12 PROGRAM 所示

```

PROGRAM CTL12(INPUT, OUTPUT);
VAR
  N, K: INTEGER;
  A, S1, S2: REAL;
BEGIN
  READLN(N);
  IF N > 0
  THEN
    BEGIN
      S1 := N / (N + 1);
      WRITELN('S1 = ' : 10:7);
      FOR K := 1 TO N DO
        BEGIN
          A := 1 / (K + 1) * K;
          S2 := S2 + A;
        END;
      WRITELN('S2 = ' : 10:7);
    ELSE
      WRITELN('ERROR IN INPUT DATA, ')
    END
  INPUT:
    OUTPUT:
    1      S1 = 0.5000000
           S2 = 0.5000000
    10     S1 = 0.9091000
           S2 = 0.9091000
    100    S1 = 0.9900990
           S2 = 0.9900995
    1000   S1 = 0.9990010
           S2 = 0.9990013
    10000  S1 = 0.9999000
           S2 = 0.9998528
    -100   ERROR IN INPUT DATA;

```

FIG CTL12 PROGRAM

这个程序运行的结果表明:

1. 如果输入的整数不是正整数, 那么就会在终端显示器上输出“输入数据错误”(ERROR IN INPUT DATA₁)。
2. 如果输入的整数是正整数, 那么 S1 和 S2 基本上是相等的。误差是由实数的运算引起的。

例4-13 通过终端键盘连续输入一系列整数。然后找出它们各自的整数商, 零的商为零。在输入负整数后不再求商。编制程序实现上述功能。

分析:

由于输入整数的个数并不确定，因此，不能用循环语句，而必须用直到语句。当然，也可以用当语句。

整数商必须一个一个地找出来。因此，可以用循环语句。求商用取模的方法。流程图如图4.10-1所示。

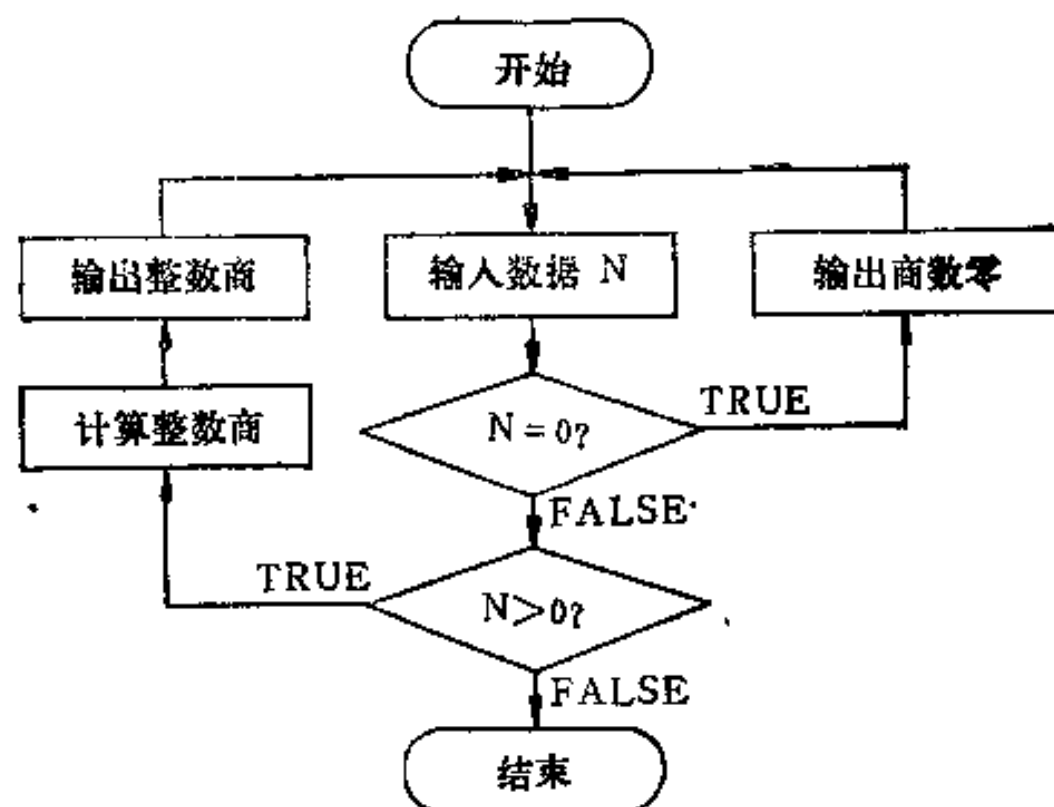


图4.10-1 例4-13的流程图

程序如 FIG CTL13 PROGRAM所示。

```

PROGRAM CTL13(INPUT, OUTPUT),
VAR
  N, D: INTEGER,
BEGIN
  REPEAT
    READ(N),
    IF N = 0
      THEN WRITELN('QUOTIENT OF 0 IS 0'),
    IF N > 0
      THEN
        BEGIN
          WRITELN('QUOTIENT OF', N:3, ':'),
          FOR D:=2 TO N DO
            IF N MOD D = 0
              THEN WRITELN(D:4)
          END
        END
      UNTIL N < 0
  END
INPUT:
20 17 0 -45 60
OUTPUT:
QUOTIENT OF 20:

```



```

4
5
19
20
QUOTIENT OF 17:
17
QUOTIENT OF 0:
0

```

FIG CTL13 PROGRAM

这个程序在执行时，尽管输入了五个整数，但是，由于输入负整数后不再求商，因此，-45和60的整数商没有计算和输出。

例4-14 连续输入 N 个整数，找出其中最大的整数和最小的整数。规定输入数在整数允许范围内。

分析：

为方便起见，首先假设：将32767作为最小的整数，-32768作为最大的整数。输入一个整数后，如果它比最小整数还要小，则把它作为最小整数。如果它比最大整数还要大，则把它作为最大整数。连续输入整数，不断进行比较，最后输出最大整数和最小整数。

流程图如图4.10-2所示。

程序如 FIG CTL14 PROGRAM 所示。

```

PROGRAM CTL14(INPUT, OUTPUT),
VAR
  M, N, I, MIN, MAX: INTEGER,
BEGIN
  READLN(N);
  MIN:=MAXINT;
  MAX:=-32768;
  FOR M:=1 TO N DO
    BEGIN
      READ(I);
      IF I<MIN
      THEN MIN:=I;
      IF I>MAX
      THEN MAX:=I;
    END;
  WRITELN(N, ' NUMBERS READ, ');
  WRITELN('THE SMALLEST NUMBER:', MIN);
  WRITELN('THE LARGEST NUMBER:', MAX);
END
INPUT
6
-6 27 101 0 -20 -99
OUTPUT:

```

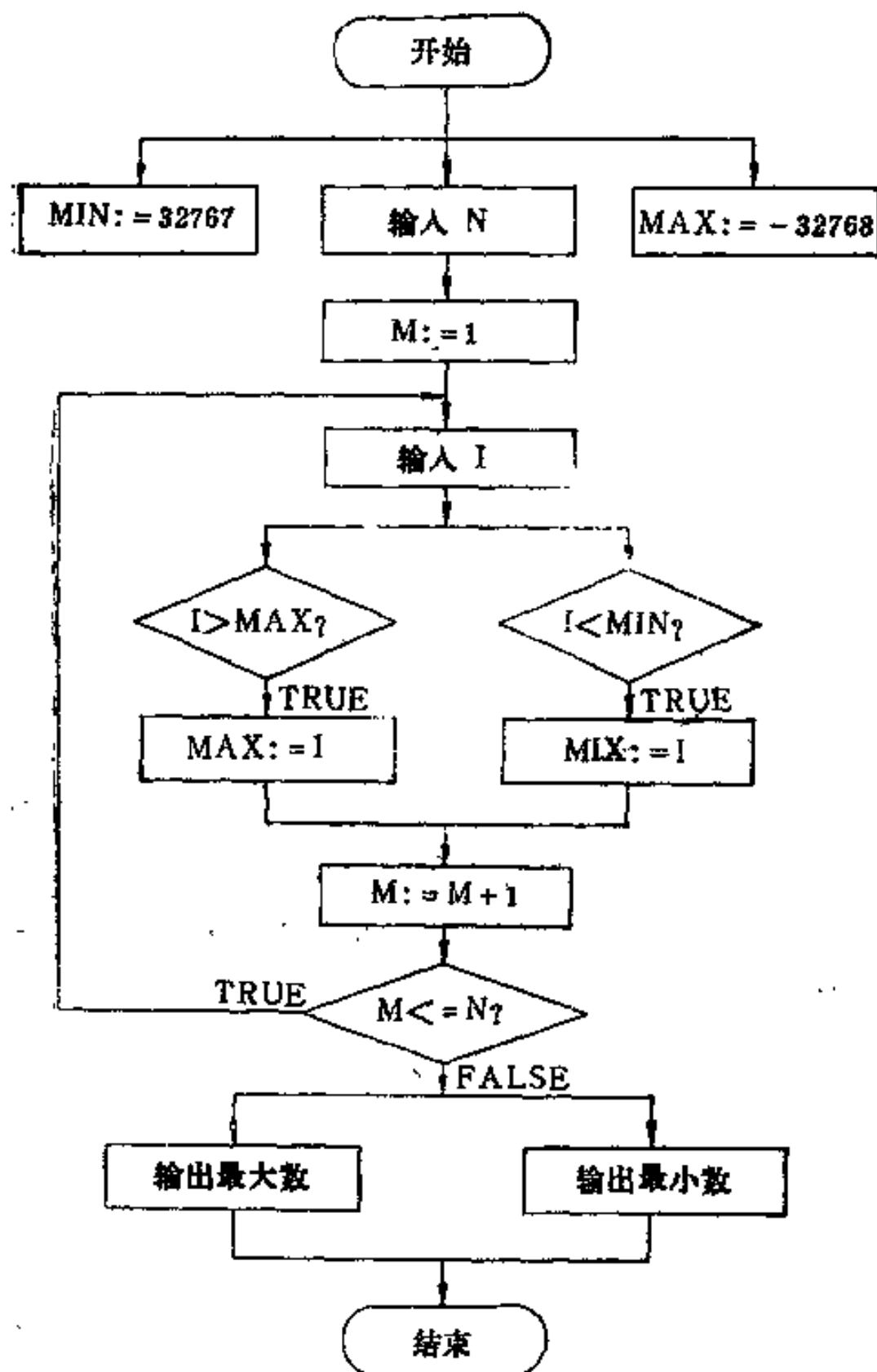


图 4.10-2 例 4-14 流程图

6 NUMBERS READ;

THE SMALLEST NUMBER: -99

THE LARGEST NUMBER: 101

INPUT:

3

-32768 0 -1

OUTPUT:

3 NUMBERS READ;

THE SMALLEST NUMBER: -32768

THE LARGEST NUMBER: 0

INPUT:

2

0 3

OUTPUT:

2 NUMBERS READ;

THE SMALLEST NUMBER: 0
 THE LARGEST NUMBER: 3
 FIG CTL14 PROGRAM

请思考:

1. 为什么不输入数时会出现最大整数为-32768, 最小整数为+32767? 能否防止这种现象发生?

2. 如果输入数超过整数允许范围, 会出现什么现象?

例4-15 用牛顿迭代法求方程的单值根, 方程为 $X^2 - 6x + 6 = 0$ 。精度根据程序员的要求确定。

分析:

解这个方程是一件容易的事, 例4-3早已解决, 它有两个不相等的实根。

牛顿迭代法(NEWTONIAN ITERATION)是一种近似方法。它通过某种极限的过程去逼近精确值, 求得满足误差要求的一个近似解, 即单值根。对于多根的方程, 它必须通过在不同区间多次迭代, 才能求出各个根。

为了有利于分析问题, 我们作函数 $y = f(x)$, $f(x) = x^2 - 6x + 6$ 。 $f(x)$ 的图形如图4.10-3所示,

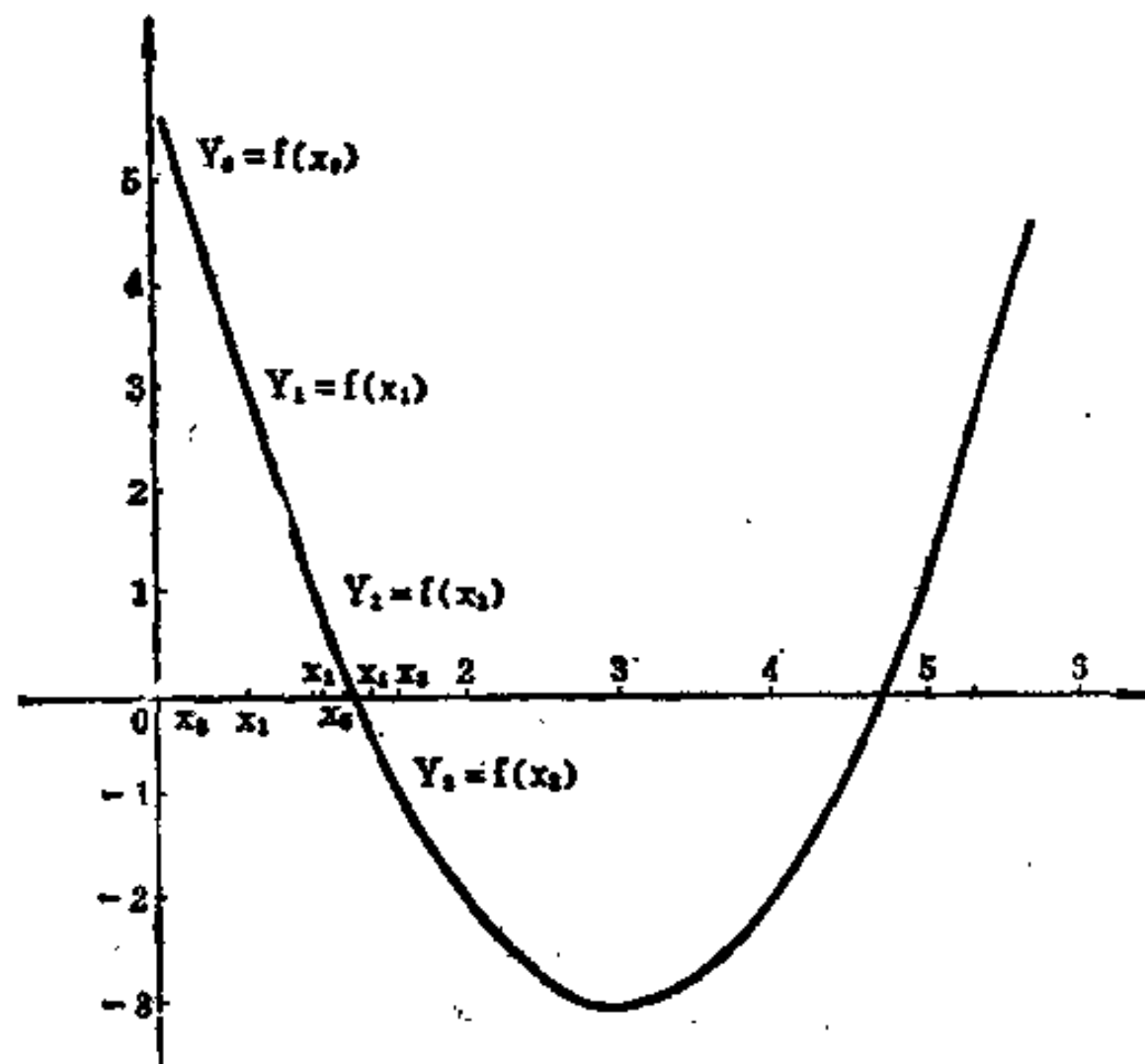


图4.10-3 函数 $f(x) = x^2 - 6x + 6$

从图形可知 $y = 0$ 有两种情况, 即方程有两个根;

一个根在 1 和 2 之间的某个 x 值;

另一个根在 4 和 5 之间的某个 x 值。

现在我们规定:

$X_0(XO)$: 给定的 X 值范围的起始值 (初值)。

$X_n(XN)$: 给定的 X 值范围的终止值 (终值)。

$\Delta X(\text{DELTA})$: X 值变化量的步长。

$\epsilon(\text{EPSILON})$: 设计的精度, 即允许误差。

很显然, 如果给定的 x 值范围不合理, 则找不到方程的根。如 $x_0=0$, $x_1=1$; 或 $x_0=2$, $x_1=4$; 或 $x_0=5$, $x_1=\text{MAXINT}$ 等。

只有给定的 x 值范围适当时, 才能求出方程的根。如 $x_0=1$, $x_1=2$; 或 $x_0=4$, $x_1=5$; 或 $x_0=1$, $x_1=6$ 都可以找出方程的根。但是一次只能找到一个根。当然 在一个更小的范围内找根会更好 更快。此如, $x_0=1.2$, $x_1=1.3$; 或 $x_0=4.7$, $x_1=4.8$ 。但是, 在未知的情況下 确定这样的范围是困难的。

牛顿迭代法见示意图图4.10-4, 它是图4.10-3局部的放大。其考虑步骤大致如下:

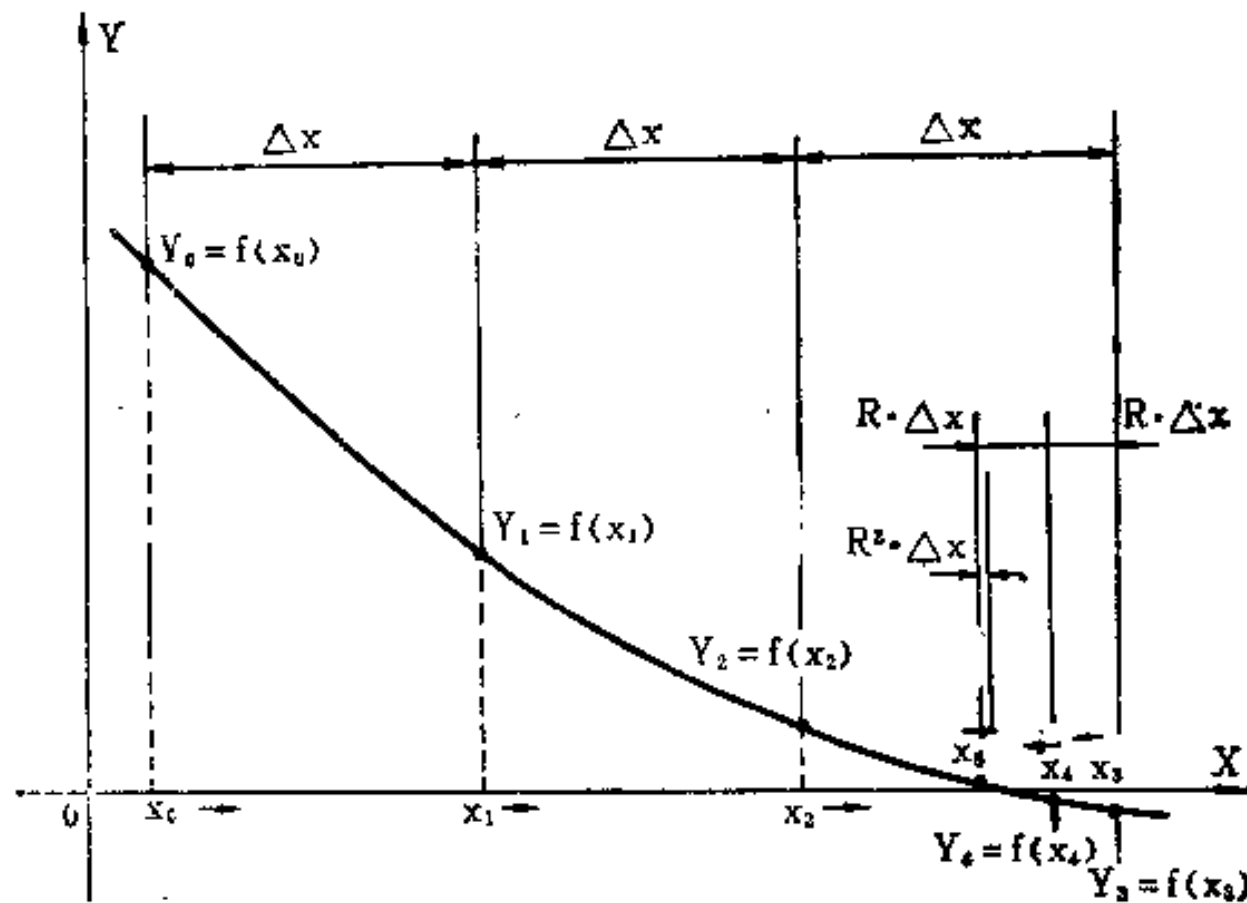


图 4.10-4 牛顿迭代法示意图

1. 根据 x_0 , 求对应的 $y_0=f(x_0)$ 。
如果 $f(x_0) \approx 0$, 在这里 $f(x_0) > 0$; 转步骤 2;
2. $x_1 := x_0 + \Delta x$, $y_1=f(x_1)$ 。
如果 $f(x_1) \approx 0$, 在这里 $f(x_1) > 0$ 转步骤 3;
3. $x_2 := x_1 + \Delta x$, $y_2=f(x_2)$ 。

依次类推, 继续判断。

4. 如果 $x_3 := x_2 + \Delta x$, $y_3=f(x_3)$, 而 $f(x_3) < 0$ 则说明函数值已反号, 在 x_2 和 x_3 之间方程有一个根。这时应缩短步长, 原来的增量 Δx 应乘一个恰当的比例因子, 比如 $R=0.1$, 则新步长为 $R \cdot \Delta x$ 。

5. $x_4 = x_3 - R \cdot \Delta x$, $y_4=f(x_4)$, $f(x_4) < 0$;

6. $x_5 = x_4 - R \cdot \Delta x$, $y_5=f(x_5)$, $f(x_5) > 0$ 。

这说明在 x_4 和 x_5 之间方程有一个根。进一步缩小步长为 $R^2 \cdot \Delta x$, 继续上述求 x_i 和 y_i 的过程。当运行到 $x = x_i$ 时, 如果步长小于允许误差时, 则取方程根为 $x_i - \frac{\Delta x}{2}$ 。

牛顿迭代法的流程图如图4.10-5所示。

例4-15的程序如FIG CTL15 PROGRAM所示。

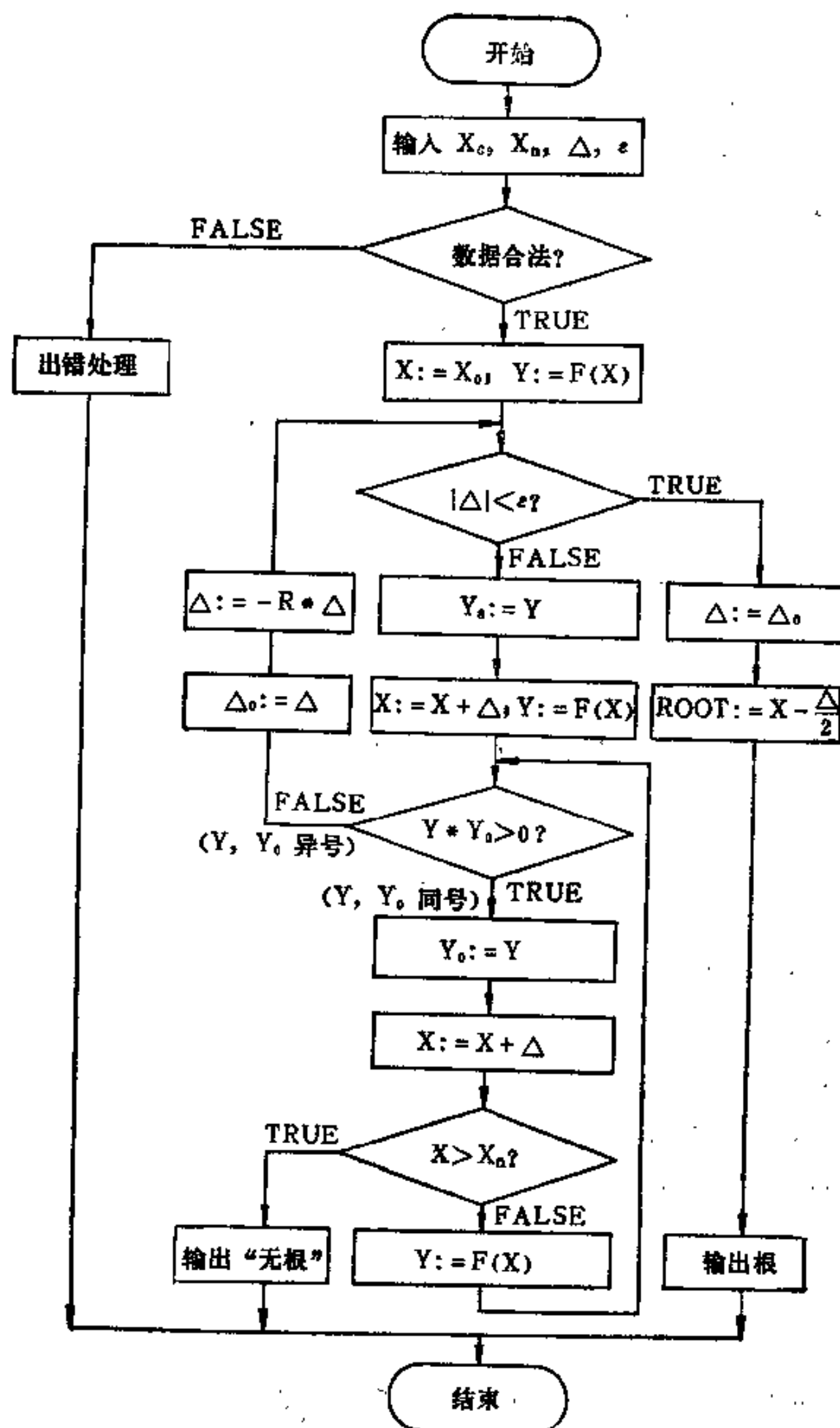


图4.10-5 牛顿迭代法流程图

```

PROGRAM CTL15(INPUT, OUTPUT),
  LABEL
    100,
  CONST
    R=0.1,
  VAR
    X0, X, XN, Y0, Y, ROOT,
    DELTA0, DELTA, EPSILON, REAL,
  BEGIN

```

```

READLN(X0, XN, DELTA, EPSILON),
IF      (X0>XN) OR (DELTA>(XN-X0))
OR (EPSILON>DELTA)
THEN WRITELN('ERROR IN INPUT DATA, ')
ELSE
  BEGIN
    X:=X0;      Y:=SQR(X)-6*X+6;
    WHILE ABS(DELTA)>=EPSILON DO
      BEGIN
        Y0:=Y;      X:=X+DELTA;
        Y:=SQR(X)-6*X+6;
        WHILE Y0*Y>0.0 DO
          BEGIN
            Y0:=Y;      X:=X+DELTA;
            IF      X>XN
            THEN
              BEGIN
                WRITELN('CANNOT FIND ROOT, '),
                GOTO 100
              END
            ELSE      Y:=SQR(X)-6*X+6
          END;
        DELTA0:=DELTA;
        DELTA:=-DELTA*R;
      END;
    DELTA:=DELTA0;
    ROOT:=X-DELTA/2.0;
    WRITELN(ROOT:8:6)
  END;
100:
END.

```

INPUT:	OUTPUT:
5.0 1.0 0.001 0.1	ERROR IN INPUT DATA,
0.0 5.0 1.0 0.001	1.267500
0.0 3.0 0.1 0.0001	1.267951
3.0 5.0 0.1 0.0001	4.732045
0.0 1.0 0.1 0.0001	CANNOT FIND ROOT,
5.0 8.0 0.1 0.0001	CANNOT FIND ROOT,

FIG CTL15 PROGRAM

例4-16 某食堂采购员带人民币100元去市场买鸡。已知每只小鸡0.5元，每只公鸡2元，每只母鸡3元。现在要求100元钱正好买100只鸡。请编程序帮助采购员制订采购方案。

分析：

假如买 I 只母鸡， J 只公鸡， K 只小鸡，则必须 $I+J+K=100$ ，而且必须花去人民币

$3I + 2J + 0.5K = 100$ 。由于有三个变量，只有两个方程，不能直接通过解三元一次方程组来确定方案，必须通过三重循环来计算。

程序如 FIG CTL16A PROGRAM 所示。

```
PROGRAM CTL16A(OUTPUT);
VAR
  I, J, K, SUM, T1:INTEGER;
  RSUM, T:REAL;
BEGIN
  T:=TIME;
  WRITELN('  I:  J:  K:');
  FOR I:=0 TO 100 DO
  FOR J:=0 TO 100 DO
  FOR K:=0 TO 100 DO
  BEGIN
    SUM:=I+J+K;
    RSUM:=3*I+2*J+0.5*K;
    IF (SUM=100) AND (RSUM=100)
      THEN WRITELN(I:5, J:5, K:5)
    END;
  T1:=ROUND(60*(TIME-T));
  WRITELN('T1 =', T1:3, 'MINUTES')
  END.
```

OUTPUT:

I:	J:	K:
2	30	88
5	25	70
8	20	72
11	15	74
14	10	76
17	5	78
20	0	80

T1 =50 MINUTES

FIG CTL16A PROGRAM

这个程序通过三重循环语句来实现，根据鸡的总数和钱的总数是否都是 100 来判断方案是否成立。

程序执行后共输出七种方案，采购员可以根据膳友的爱好的去采购。

这个程序用了函数 TIME，TIME 是 OMSI PASCAL-1 语言扩展的一种标准函数。它表示当天的一个时间，函数值是实数，以小时为单位。如果上机时通过键盘命令输入了初始时间，如

.TIME 7:15:00

则在机器中将 TIME 立即转换成实数 7.25。然后经时钟不断计时。

如果上机时没有输入初始时间，则默认为 0.0。

在这个程序中，先将目前的时间 (TIME) 赋值给变量 T，然后执行所要求的计算，最后，计算出解决这个问题所花费的时间，即 (TIME - T)，乘60是为了以分为单位，舍入函数是为了获得整分。

执行这个程序共花费了50分钟时间，这是很大的浪费。原因在于它必须进行 100 万次循环，即 $100 \times 100 \times 100$ 次。

如何节约时间呢？只要稍加考虑就会发现，100元钱绝对不可能买100只母鸡或100只公鸡。事实上，最多超不过30只母鸡或50只公鸡。因此，我们将程序稍加修改，形成新程序如 FIG CTL16B PROGRAM所示。

```

PROGRAM CTL16B(OUTPUT);
VAR
  I, J, K, SUM, T2: INTEGER;
  RSUM, T: REAL;
BEGIN
  T := TIME;
  WRITELN(' I: J: K: ');
  FOR I := 0 TO 30 DO
    FOR J := 0 TO 50 - I DO
      FOR K := 0 TO 100 - I - J DO
        BEGIN
          SUM := I + J + K;
          RSUM := 3 * I + 2 * J + 0.5 * K;
          IF (SUM = 100) AND (RSUM = 100)
            THEN WRITELN(I:5, J:5, K:5)
        END;
      T2 := ROUND(60 * (TIME - T));
      WRITELN('T2 = ', T2:3, 'MINUTES')
    END;
  END;
OUTPUT:
  I: J: K:
  2 30 68
  5 25 70
  8 20 72
  11 15 74
  14 10 76
  17 5 78
  20 0 80
  T2 = 4 MINUTES

```

FIG CTL16B PROGRAM

执行这个新程序花费 4 分钟时间。节约46分钟。实际上，提高程序技巧，考虑得更仔细一些，可以节约更多的时间。见程序 FIG CTL16C PROGRAM。该程序仅仅用 3 秒钟。

```

PROGRAM CTL16C(OUTPUT);
VAR

```



```

      I, J, SUM, T2:INTEGER,
      RSUM, T:REAL,
BEGIN
      T:=TIME,
      Writeln('      I:      J:      K:'),
      FOR I:=0 TO 30 DO
      FOR J:=0 TO 50-1 DO
      IF (3*I+2*J+0.5*(100-I-J))=100
      THEN Writeln(I:5, J:5, (100-I-J):5),
      T2:=TRUNC(3600*(TIME-T)),
      Writeln('T2=', T2:2, ' SECONDS')
END.

```

OUTPUT:

```

      I:      J:      K:
      2      30      68
      5      25      70
      8      20      72
     11      15      74
     14      10      76
     17       5      78
     20       0      80

```

T2=3 SECONDS

FIG CTL16C PROGRAM

这个例题说明了一点，设计程序时必须注意质量。高质量的程序必须功能齐全，尽量节省内存空间和运行时间。

例4-17 输入奇数 A ，计算并输出 A 位的魔方阵。

分析。

什么叫魔方阵？

魔方阵就是 A^2 个不同的正整数按方阵形排列时，它的每一行，每一列以及沿对角线的几个数的和具有同一性质的方阵。由 1 到 A^2 个自然数构成的魔方阵是最基本的魔方阵。

魔方阵的常数为沿每一行，每一列以及沿对角线的几个数的和。这个常数为：

$$\frac{1}{2}A(A^2+1)$$

例如，自然数 1 到 9 可以排成三行三列，每行每列或对角线之和为 $\frac{1}{2} \times 3(3^2+1)=15$ 。

魔方阵如右图。

如何确定这些自然数在方阵中的位置呢？

首先确定 1 的位置。通常，1 总是在第一行最中间的位置。

其次确定其它自然数的位置。一般地说，目前自然数的右上方是下一个自然数的位置。例如：

4 的右上方是 5，5 的右上方是 6。

8	1	6
3	5	7
9	4	2

如果目前自然数在第一行，但不在最右侧，则下一个自然数在最后一行，列数右移一列。

例如：

1 的右“上”方是 2，8 的右“上”方是 9。由于“上方”不存在，转到最后一行。

如果目前自然数在第一行的最右侧，则下一个自然数在目前自然数的下侧。例如：

6 的下侧是 7。

如果目前自然数在其它行的最右侧，则下一个自然数在上一行的最左侧。例如：

2 的上一行的最左侧为 3；

7 的上一行的最左侧为 8。

最后，按照矩阵输出的方式输出这个魔方阵。

上面的分析仅是一种形象化的描述。其流程图如图 4.10-6 所示。

例 4-17 的程序，如 FIG CTL17 PROGRAM 所示。

```
PROGRAM CTL17(INPUT, OUTPUT);
LABEL
  10;
TYPE
  MATRIX=ARRAY (1..15, 1..15) OF INTEGER;
VAR
  M      :MATRIX;
  I, J, K, A, C:INTEGER;
BEGIN
  (• PART 1.....INPUT DATA •)
  READLN(A);
  IF (A>0) AND (A MOD 2 <>0)
  THEN WRITELN('A=', A:3)
  ELSE
    BEGIN
      WRITELN('ERROR IN INPUT DATA!');
      GOTO 10
    END;
  (• PART 2.....COUNT MATRIX •)
  WRITELN;
  C:=SQR(A);      J:=1;
  K:=(A+1) DIV 2;
  FOR I:=1 TO C DO
    BEGIN
      M [J, K] :=I;
      IF I MOD A=0
      THEN
        IF J=A
        THEN J:=1
        ELSE J:=J+1
```

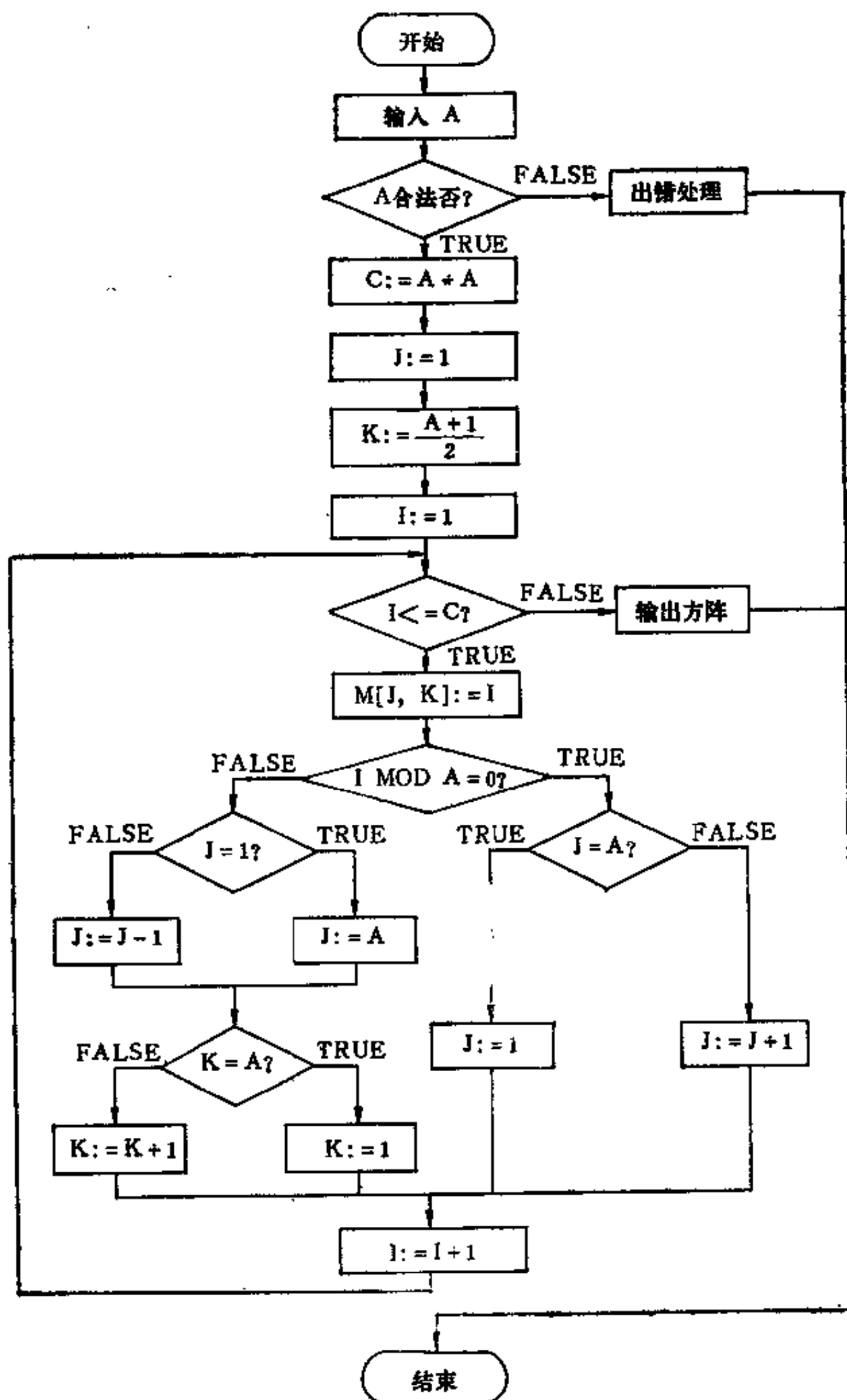


图4.10-8 魔方阵的流程图

```

ELSE
  BEGIN
    IF J=1
      THEN J:=A
      ELSE J:=J-1,
    IF K=A

```

```

        THEN K:=1
        ELSE K:=K+1;
    END
END
(• PART 3.....OUTPUT MATRIX •)
FOR I:=1 TO A DO
    BEGIN
        FOR J:=1 TO A DO WRITE ((M [I, J] ):4);
        WRITELN
    END;

```

10:

END.

INPUT:

8

OUTPUT:

A= 3

8 1 6

3 5 7

4 9 2

INPUT:

6

OUTPUT:

A= 6

17 24 1 8 15

23 5 7 14 10

4 6 13 20 22

10 12 19 21 3

11 18 25 2 9

INPUT:

--3

OUTPUT:

ERROR IN INPUT DATA;

INPUT:

10

OUTPUT:

ERROR IN INPUT DATA;

INPUT:

17

OUTPUT:

ARRAY BOUNDS ERROR

FIG CTL17 PROGRAM

这个程序的执行部分有三部分组成。

第一部分，输入数据，判断其是否合法。当输入负数或偶数时作为“输入数据错误”处

理，结束程序的运行。

第二部分，魔方阵的建立过程。首先将 1 放到第一行的中间，然后根据目前自然数 1 的特点（是否为 A 的倍数）及其坐标（J 和 K）确定下一个自然数（i + 1）的坐标（J 及 K）。

第三部分，通过二重循环输出魔方阵。

必须指出，本程序允许的最大方阵为 15 行和 15 列。当 $A > 15$ 时，程序执行时发生“数组越界错误”（ARRAY BOUNDS ERROR）。

例 4-18 输入本世纪（从 1900 年开始）及下一世纪内任何一天的年、月、日，经过计算自动输出那一天是星期几。要求星期几用英文表示。请编制程序完成上述功能。

分析：

根据题意，程序大致必须由三部分组成：输入年、月、日；计算星期几；输出星期几。

1. 输入年月日。

输入年月日用读语句完成。

如果输入的年份超过题意的要求，输入的月日不符合历法的规定，则作为“输入数据错误”处理。

2. 计算星期几。

这是解决本题的关键。

首先，根据天文历法的规定：每 400 年中（例如 1701 年至 2100 年）只能有 97 个闰年。

凡是能被 100 整除、且能被 400 整除的年份方为闰年。如 1600 年，2000 年；

凡是不能被 100 整除、但能被 4 整除的年份为闰年。如 1980 年，1996 年。

其余的年份都是平年，包括能被 100 整除、但不能被 400 整除的年份。如 1800 年，1900 年，2100 年。

闰年的二月是 29 天，平年的二月是 28 天。

其次，计算总天数受到整数范围的限制。考虑到平年 365 天，52 个星期多一天。闰年 366 天，52 个星期多二天。 $365 \text{ MOD } 7$ 与 $1 \text{ MOD } 7$ 效果相同， $366 \text{ MOD } 7$ 与 $2 \text{ MOD } 7$ 效果等同。为简化起见，可认为平年累积天数为 1，闰年累积天数为 2，即平年累积天数 1 再加 1 即可。为了不影响计算星期几，可以从 1900 年 1 月 1 日开始相对地计算总天数。其办法如下：

$$\text{总天数} = \text{平年累积值} + \text{闰年累积值} + \text{月累积值} + \text{日期}$$

即

$$\text{DSUM} := (\text{YEAR} - 1901) + (\text{YEAR} - 1901) \text{ DIV } 4 + \text{D1} + \text{DAY};$$

最后，计算星期几，考虑到 1899 年 12 月 31 日为星期一，则初值 $\text{WEEK} := 1$ 。

某一天是星期几则将 WEEK 初值加总天数之和对 7 取模，模数就是星期几。

3. 输出星期几。

这是很简单的。用情况语句就可以。

框图如图 4.10-7 所示。

例 4-18 的程序，如 FIG CTL18 PROGRAM 所示。

```
PROGRAM CTL18 (INPUT, OUTPUT),  
LABEL
```

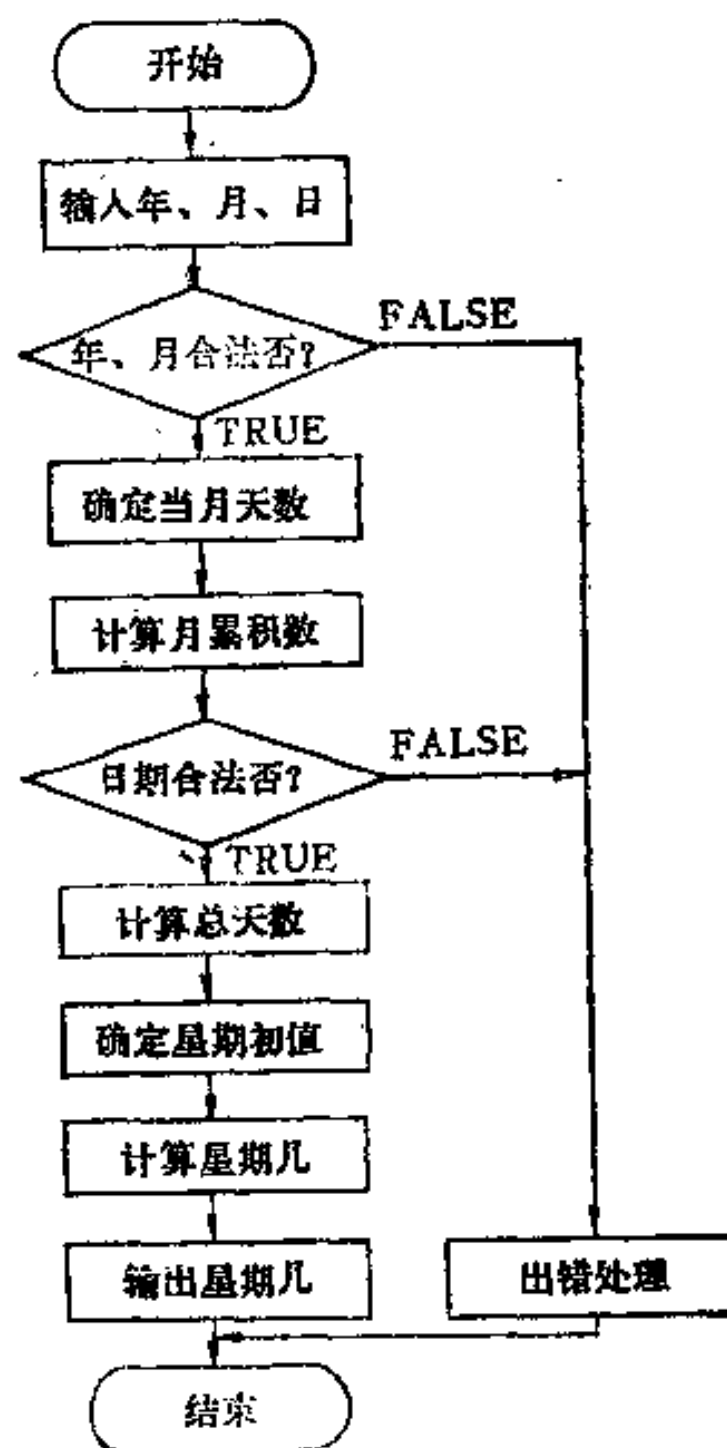


图 4.10-7 例4-18流程图

```

100;
VAR
  YEAR, MONTH, DAY, WEEK: INTEGER,
  DSUM, M2, D0, D1: INTEGER,
BEGIN
  (• PART 1.....INPUT DATA •)
  READLN (YEAR, MONTH, DAY);
  IF (YEAR<1901) OR (YEAR>2100) OR
    (MONTH<1) OR (MONTH>12)
  THEN
    BEGIN
      WRITELN('ERROR IN INPUT DATA, ');
      GOTO 100
    END;
  (• PART 2.....COUNT WEEK •)
  IF (YEAR MOD 100<>0) AND (YEAR MOD 4=0)
    OR (YEAR MOD 100=0) AND (YEAR MOD 400=0)
  THEN M2:=29

```

```

ELSE M2:=28;
CASE MONTH OF
  1,
    BEGIN
      D0:=31; D1:=0
    END;
  2,
    BEGIN
      D0:=M2; D1:=31
    END;
  3,
    BEGIN
      D0:=31; D1:=M2+31
    END;
  4,
    BEGIN
      D0:=30; D1:=M2+62
    END;
  5,
    BEGIN
      D0:=31; D1:=M2+92
    END;
  6,
    BEGIN
      D0:=30; D1:=M2+123
    END;
  7,
    BEGIN
      D0:=31; D1:=M2+153
    END;
  8,
    BEGIN
      D0:=31; D1:=M2+184
    END;
  9,
    BEGIN
      D0:=30; D1:=M2+215
    END;
  10,
    BEGIN
      D0:=31; D1:=M2+245
    END;
  11,
    BEGIN

```

```

        D0:=30; D1:=M2+270
    END;
12;
    BEGIN
        D0:=31; D1:=M2+306
    END;
END;
IF (DAY>D0) OR (DAY<1)
THEN
    BEGIN
        WRITELN('ERROR IN INPUT DATA: ');
        GOTO 100
    END;
DSUM:=(YEAR-1901)+(YEAR-1901) DIV 4+D1+DAY;
WEEK:=1;
WEEK:=(WEEK+DSUM) MOD 7;
(* PART 3-----OUTPUT WEEK *)
CASE WEEK OF
    0, WRITELN('SUNDAY');
    1, WRITELN('MONDAY');
    2, WRITELN('TUESDAY');
    3, WRITELN('WEDNESDAY');
    4, WRITELN('THURSDAY');
    5, WRITELN('FRIDAY');
    6, WRITELN('SATURDAY');
END;
100;
END.

```

INPUT,			OUTPUT,
1901	1	1	TUESDAY
1911	10	10	TUESDAY
1949	10	1	SATURDAY
1974	6	21	FRIDAY
1976	1	8	THURSDAY
1976	9	9	THURSDAY
1981	2	21	SATURDAY
1871	3	18	ERROR IN INPUT DATA;
1981	2	30	ERROR IN INPUT DATA;

FIG CTL18 PROGRAM

这个程序运行的结果表明，它是符合实际情况的。只要年、月、日输出都符合历法和程序的规定，它会输出星期几。只要年、月、日中有一项不符合历法规定或题目要求，它就指出“输入数据错误”。如1871年3月18日是存在的，符合历法规定，但不符合题目要求。如1981年2月30日是不存在的，它不符合历法规定。

这个程序的核心语句如下:

DSUM:= (YEAR-1901)+ (YEAR-1901) DIV 4 + D1 + DAY;

WEEK:= (WEEK + DSUM) MOD 7;

请思考并回答如下问题:

1. YEAR-1901代表什么? 为什么不用 (YEAR-1901) * 365?
2. (YEAR-1901) DIV 4 代表什么?
3. D1和DAY 分别代表什么?
4. 赋值语句后面的WEEK代表什么?
5. MOD 7 表示什么意义?

本章小结

本章介绍了PASCAL语言常用的基本语句,是本书语句介绍的中心。

在这些控制流程的语句中,重复语句用途最广。这是由于计算机运算速度很快,善于承担繁杂的重复性的工作。因此,重复语句是重点。三种重复语句(当语句,直到语句和循环语句)中,重点介绍了循环语句。其中有循环语句的基本概念,多重循环,退出循环及循环转移等。基本思想同样适用于当语句和直到语句。希望读者在编制程序时根据各自的特点灵活地应用。

三种重复语句的共同点在于都要重复执行某些语句,这些语句形成所谓“循环体”,这些程序为循环结构。不同点是不同的重复语句控制重复次数的时机和办法不一样。一般地说,当语句和直到语句的重复次数依赖于循环体的执行情况,而循环语句的重复次数独立于循环体的执行情况,在编写程序时,就已经确定了重复次数。在判断条件与执行循环体的顺序上,当语句和循环语句是首先判断条件,然后执行循环体。循环可能一次也不执行。而直到语句是首先执行循环体,然后判断条件,至少执行一次循环体。

两种条件语句(如果语句和情况语句)在程序中应用广泛,它们的共同点是根据具体的情况执行相应的语句。如果语句是情况语句的特殊情况,情况语句是如果语句的扩展情况,它们组成的程序结构是分支结构。

转移语句和退出语句也是经常用到的。它们的共同点都是从某一个地方转到另一个地方。使用时往往跟条件语句相结合。它们的区别在于退出语句只是从循环语句中转移到下一语句,而转移语句的适用范围更广。当然,在嵌套的情况下,使用转移语句要注意层次。

在本章中各有关章节,一再提到可能出现的死循环问题及可能的原因,应该在应用重复语句与转移语句中引起足够的重视,免得为此在人工与机时上化费过高的代价。

本章习题

1. 编制程序计算。

$$y = \begin{cases} \cos(x+3.0) & 0 \leq x < 10 \\ \cos^2(x+7.5) & 10 \leq x < 20 \\ \cos^4(x+4.3) & 20 \leq x < 30 \end{cases}$$

$$z = \begin{cases} \sin(2x+4) & 0 \leq x < 10 \\ \sin^3(2x+6) & 10 \leq x < 20 \\ \sin^5(2x+8) & 20 \leq x < 30 \end{cases}$$

输入 x ，计算并输出 y 和 z 。

2. 输入10个整数，按从大到小的顺序将其排列并输出排列的结果。

3. 计算 $M = 1^1 + 2^2 + 3^3 + 4^4 + \dots + N^N$ ，直到 $M \geq 10^{10}$ ， M 为实数。

4. 计算 $S = 1 + \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N}$ ，直到最后一项小于 10^{-6} 。

5. 计算并输出

$$S_1 = \sum_{n=1}^5 n! \quad \text{和} \quad S_2 = \frac{1}{15} \sum_{n=1}^{15} \frac{n}{n+3} x^{2n}$$

其中 $x = 1.25$ 。

6. 用牛顿叠代法求一元三次方程的根。

$$方程为 \quad 4 + 16.7x + 9.2x^2 - 1.02x^3 = 0$$

方程根近似值满足 $|x_{n+1} - x_n| \leq 10^{-5}$

7. 鸡兔同笼，计数共有头 M 个，脚 N 只。编制程序计算鸡兔各为多少？其中 M 和 N 由终端输入。

8. 100匹马驮100担货，大马一匹驮3担，中马一匹驮2担，小马两匹驮一担货。试编程序计算大、中、小马的数目。

试从几种不同的方案中找出程序运行最省的来。

9. 从下面的程序及其运行结果，分析产生这一结果的原因，进一步体会无条件转移语句作为构造型语句的成分语句的不宜之处。

```
PROGRAM CONTROL (OUTPUT);
LABEL 100;
VAR I, J, INTEGER;
BEGIN
  FOR I:=1 TO 10 DO
  BEGIN
    J:=SQR(I);
    IF J=9 THEN GOTO 100;
    WRITELN ('I=', I:2);
  100 WRITELN ('BY THE END OF PROGRAM, I=', I:2)
  END
END
OUTPUT,
I = 1
I = 2
BY THE END OF PROGRAM, I = 3
```

10. 编一个程序，把从终端上输入的一串连续的数字符转移成相应的整数值，并把这一数值用八进制形式输出。

第五章 过程 和 函 数

设计一个比较复杂的程序时，首先，按照程序中要实现的若干主要功能，往往将程序划分为几个相对独立的部分，通常称它们为子程序或分程序。其次，要确定子程序与主程序、子程序与子程序之间的关系。最后，仔细地编制程序，进行调试和运行。当然，设计一个子程序时，如果子程序比较大，也必须先粗后细，分块处理。

在这样设计程序时，每一步之间都是密切联系的。前一步是后一步的前提与出发点，后一步是前一步的补充与完善。这样设计出来的程序，各部分之间的逻辑关系明确，结构清晰，易读易懂，便于查错和修改。这种设计方法通常属于结构式程序设计方法。它帮助用户得到正确、可靠和高效率的程序。

定义和调用过程是程序语言的主要特征之一。过程是模块程序设计（又叫结构式程序设计）的主要手段，同时也是节省程序代码和扩充语言能力的主要途径。

一个过程一旦定义之后就可以在别的地方调用它。调用与被调用（过程）之间的信息往来一般通过全程变量或参数的传递来实现。

PASCAL 语言是实现结构式程序设计的比较理想的一种语言。它所使用的主要手段之一，就是过程和函数。这就是本章所要讨论的问题。

本章涉及的基本概念比较多，对于一些初学者，尤其是对那些不太熟悉计算机及程序设计的读者，要准确地掌握这些概念，熟练地应用它们，会有个逐步适应的过程，开始时可能会感到困难一些。为此本章特别注意在内容编写上由浅入深，深入浅出，尽量多举例题来说明。此外，对于用 * 标志的部分，初学者可以暂不阅读。

§1. 标准过程和标准函数

过程和函数就是通常所说的子程序。在进行程序设计时，往往会用到许多过程和函数。例如，从终端键盘输入数据时，用读语句。从终端显示器输出数据时，用写语句。读语句和写语句实际上都是过程。又如，计算 $0^{\circ} \sim 359^{\circ}$ 每一度的正弦值就要用函数 $\text{SIN}(x)$ 等等。这些过程和函数，都是由编译程序事先准备好的。只要遵照一定的调用规则使用就行了。通常把这些过程或函数叫做标准过程或标准函数。

例5-1 输入一个整数，计算并输出其正弦函数值。

程序如 FIG PF01 PROGRAM 所示。

```
PROGRAM PF01 (INPUT, OUTPUT);
VAR
  X: INTEGER;
  Y: REAL;
BEGIN
  READLN(X);
  Y:=SIN (X);
```

```

WRITELN(Y)
END.
FIG PF01 PROGRAM

```

在这个程序中,用到了过程 READLN和 WRITELN,还用了函数SIN。使用这些标准过程函数,使程序书写简单,结构清晰,易读易懂。

如果在编译程序中不提供这些标准过程与标准函数,输入变量,计算函数值以及输出变量或计算结果的具体程序,都由用户自己去编写,那就不可想象。这样必然会把绝大多数用户拒之门外。

由此可知,标准过程和标准函数会给用户带来了很大的方便。正确与合理地使用编译程序提供的标准过程和标准函数,是有效地进行程序设计的一个重要方面。

标准 PASCAL 语言提供的标准过程如下:

```

DISPOSE,      GET,      NEW,      PACK,
PAGE,         PUT,      READ,     READLN,
RESET,        REWRITE,  UNPACK,   WRITE,
WRITELN

```

OMSI PASCAL-1语言版本中,没有 PACK 和 UNPACK 两个过程;但是增加了 BREAK, CLOSE和EXIT 三个过程,并且扩展了 RESET 和 REWRITE两个过程的功能。

标准 PASCAL 语言提供的标准函数如下:

```

ABS,          ARCTAN,    COS,
EOF,          EOLN,      EXP,
LN,           ODD,       ORD,
PRED,         ROUND,     SIN,
SQR,          SQRT,      SUCC,
TRUNC.

```

OMSI PASCAL-1语言版本中,增加了 EXP10, LOG 和 TIME 三个函数。

上面列举的标准过程和标准函数,有的已介绍并使用过,其余的在后续章节中讨论。

§2. 简单过程和简单函数

标准过程和标准函数是由编译程序提供的最常用的过程和函数。尽管它们为用户带来了方便,但是,仅仅依靠它们还是不能满足用户的实际需要。

PASCAL 语言允许用户根据实际需要,在编制程序时定义并调用自己编制的过程和函数。在程序设计时,习惯上将这类过程和函数称为用户自定义的过程和函数。本章所讨论的正是这些过程和函数。在后续章节中,如果不加特殊说明,术语“过程”和“函数”都是指的用户定义的过程和函数。

如果用户在程序中使用自己定义的过程或函数,则必须做两件事:

第一,在程序的说明部分进行过程说明或函数说明;

第二,在程序的执行部分调用过程或函数。

在一般情况下,它们必须满足先说明,后调用的规则。特殊情况下允许向前调用。当然,

上页

只说明不使用也允许，但这是不必要的浪费。

下面看一个简单的过程和函数应用的例子。

例5-2 输入三个整数，进行过程调用和函数计算，并输出运算的结果。

程序如 FIG PF02 PROGRAM所示。

```
PROGRAM PF02 (INPUT, OUTPUT),
VAR
  I, J, K:INTEGER;
PROCEDURE P;
BEGIN
  I:=I+100
END;
FUNCTION F (X:INTEGER):INTEGER;
BEGIN
  F:=X+K
END;
BEGIN (* MAIN PROGRAM *)
  READLN (I, J, K);
  P;
  J:=F(I);
  K:=SQR(K);
  WRITELN (I:6, J:6, K:6)
END.

INPUT:
      1      2      3
OUTPUT:
      101     104      9
      FIG PF02 PROGRAM
```

在这个程序的说明部分，定义了三个整数类型变量 I，J 和 K。说明了一个过程 P，它的功能是将 I 加上 100 后再送给 I。说明了一个函数 F，这个函数的自变量为 X，其功能是自变量 X 加上 K 作为函数值 F，自变量和函数值都属于整数类型。

在这个程序的执行部分，共有五个语句：

语句 1 为标准过程语句 READLN，用户从终端键盘输入三个整数，如 1，2，3

语句 2 为过程语句 P，表示调用过程 P，该过程的功能用于改变变量 I 的数值。I:=I+100，执行过程 P 的结果使 I 的值为 101

语句 3 为赋值语句，将函数 F 的值赋给变量 J，达到改变 J 的目的。另外，F(I) 表示调用函数 F，在调用过程中将 I 的值自动传递给函数的自变量 X，即 $X = I = 101$ ，然后将 $(X + K)$ 的值赋给 F，即 $101 + 3 = 104$ 作为函数值，并进一步赋给变量 J，从而使 J 为 104。

语句 4 为赋值语句，将 K 的平方值作为变量 K 这个内存单元的内容。SQR 是一个标准函数。 $K := SQR(K)$ ，结果为 9。

语句 5 为标准过程语句 WRITELN。它输出经过计算后的 I，J 和 K 的值。在输入 1，2 和 3 的情况下，输出 101，104 和 9。

§3. 过程说明和过程语句

上一节的例题中使用的过程P是一个简单的过程。过程说明的一般结构如图5.3-1。

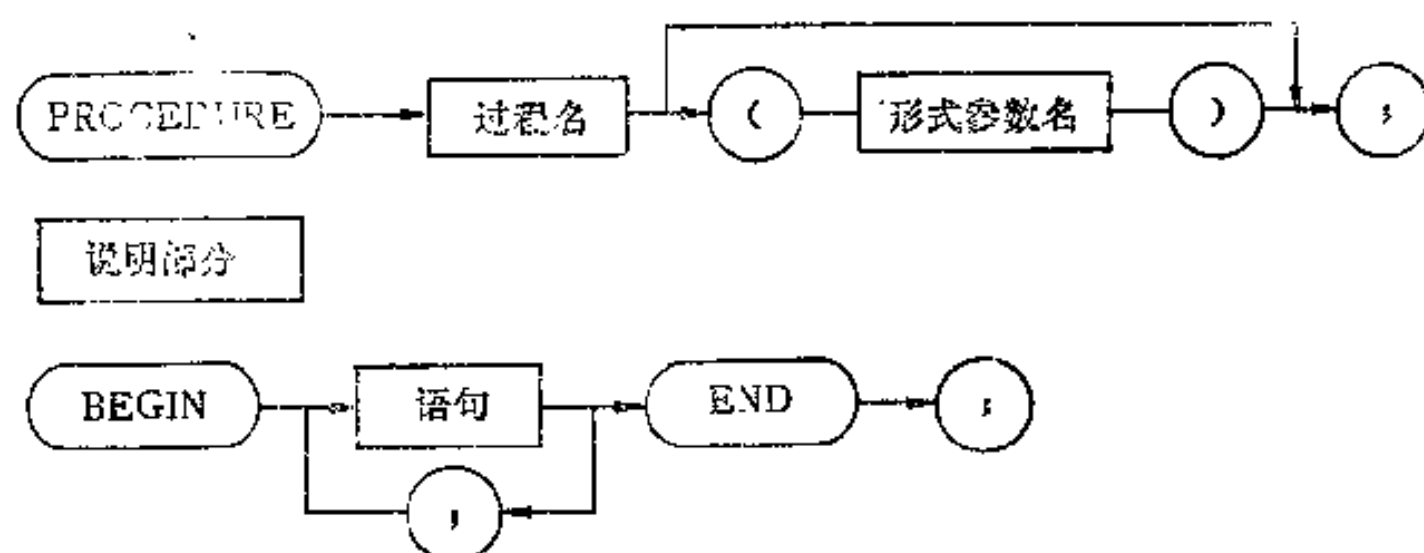


图 5.3-1 过程说明的一般结构

从图示的语法结构上看，它与第一章介绍的 PASCAL 语言的程序结构几乎完全一样。PASCAL 语言的程序结构简化如图5.3-2所示。

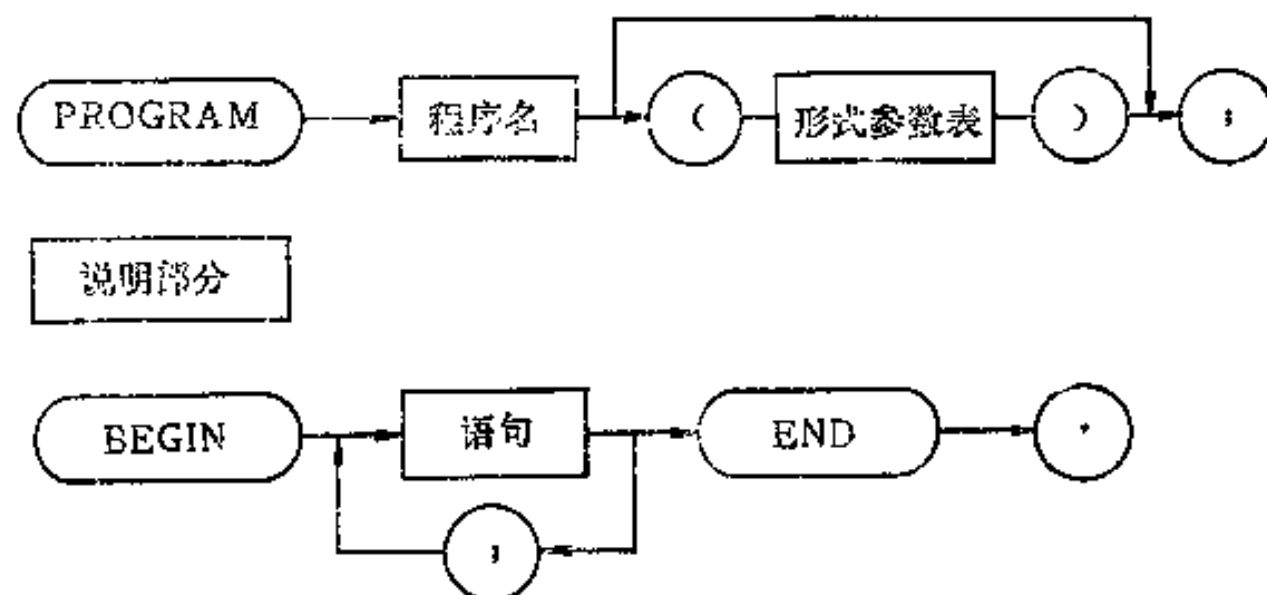


图 5.3-2 程序结构简化图

其中：

PROCEDURE 是过程说明的标志，它是 PASCAL 语言的保留关键字。每一个过程说明都必须有一个标志 PROCEDURE，不允许几个过程说明共用一个 PROCEDURE。

过程名由用户自己定义，但必须符合用户标识符的规定。过程调用时就用这个过程名。

形式参数表是过程（子程序）与主程序联系的纽带。

过程说明部分与执行部分，与程序的说明部分和执行部分的意义大致相同。区别在于前者仅仅隶属于过程。过程可以有自己的标号说明，常量定义，类型定义，变量说明，还可以有自己的过程和函数说明。过程中间套过程，简称为嵌套。

程序结构与过程结构的区别在于：

1. 标志不同。

程序的标志是保留关键字 PROGRAM。过程的标志是 PROCEDURE。

2. 参数不同。

程序的形式参数一般是文件类型的变量，如 INPUT, OUTPUT。它表示程序与外界之间联系的渠道。

过程的形式参数表示过程与主程序之间联系的纽带。它可以是数值、变量、过程或函数。当然，这些数值、变量或函数都必须属于某种数据类型。

3. 结尾不同

程序的结尾是英文句号“.”，表示程序到此结束。过程是程序的一个组成部分，用分号“;”表示这个过程结束。

4. 运行不同

程序的运行直接受操作系统控制，而过程的运行则由程序调用。程序可以调用过程，一般用户定义的过程是不能调用程序的。

简单的过程可以没有自己的说明部分，甚至没有形式参数表。如例题中的过程P。

过程说明在程序的说明部分。过程语句，又叫过程调用语句，在程序的执行部分。过程语句的一般格式如图5.3-3；

其中：

过程名必须在程序说明部分加以说明，实际参数表对应于过程说明中的形式参数表。简单的过程语句可以没有实际参数。

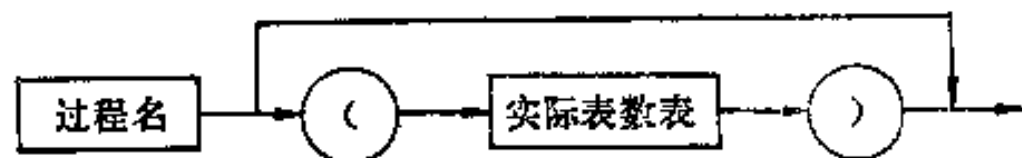


图 5.3-3 过程语句一般格式（图中“表”字因为“参”）

过程语句的作用是，在程序运行时遇到过程语句时，就转去执行相应的过程，即执行过程说明的执行部分。执行完之后，返回执行过程语句的下一个语句。过程语句中的实际参数为相应的过程提供某些“原始”数据。

如果将 PASCAL 语言与汇编语言作一简单的对照，就会发现：过程说明相当于子程序；过程语句相当于转子指令；转移语句相当于转移指令。

例5-3 编制一个程序，其中有计算平方根值的过程。

程序如 FIG PF03 PROGRAM 所示：

```

PROGRAM PF03(OUTPUT),
VAR
  I, J:INTEGER;
  R :REAL;
PROCEDURE A (X:INTEGER),
BEGIN
  IF X >= 0
  THEN
    BEGIN
      R:=SQRT(X);
      WRITELN('R=', R:7:3)
    END
  ELSE WRITELN('SQRT OF NEGATIVE IS ILLEGAL:');
END;
BEGIN (• MAIN PROGRAM •)
  I:=2;
  J:=3;
  A(I);
  A(J);

```

```

        A(0),
        A(-8)
    END
OUTPUT,
R=1.414
R=1.732
R=0.000
SQRT OF NEGATIVE IS ILLEGAL ,
    FIG PF03    PROGRAM

```

这个程序的功能很简单，就是计算 2，3，0 和 -8 的平方根。

这个程序的特点很明显，就是多次调用过程 A。过程 A 有助于理解形式参数和实际参数的关系。

在过程 A 的说明中，有一个整数类型的形式参数 X。最简单的参数表是数值参数表。

一个过程说明，可以有多种类型的参数，每一种参数类型又可以有几个形式参数。最简单的是一个某种数据的形式参数。如例 5-3 中的过程 A 就是这种情况。

关于过程形式参数的一般格式，如图 5.3-4 所示。

其中：

形式参数名由用户按照用户标识符的规定去定义；

参数类型必须是标准的数据类型或已经定义的其它数据类型。

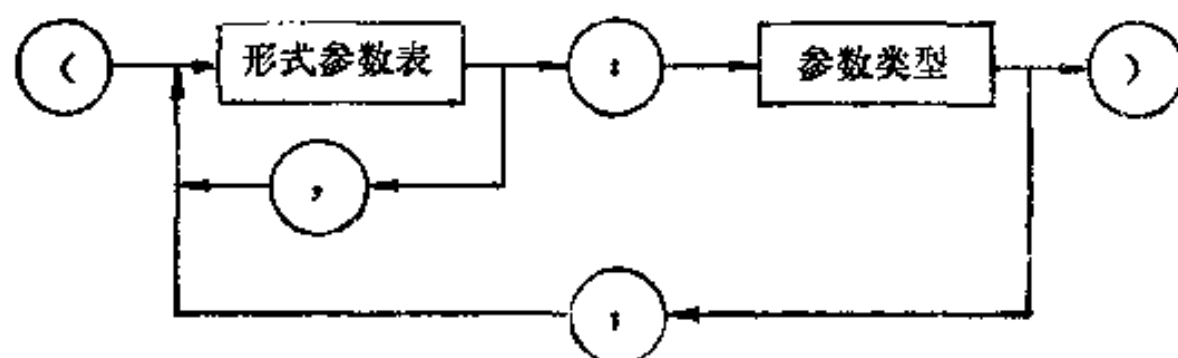


图 5.3-4 形式参数的一般格式

例如，下列过程说明是允许的：

```

PROCEDURE B (X:INTEGER, Y:BOOLEAN);
PROCEDURE C (X1, X2:INTEGER; Z1, Z2:BOOLEAN);
.
.
.

```

因为 INTEGER, REAL, BOOLEAN 都是标准的数据类型。

过程 A 的功能是在 X 为负数时指出错误性质为“负数不能开平方” (SQRT OF NEGATIVE IS ILLEGAL)，否则，计算并输出 X 的平方根 R。

在程序的执行部分，三次调用过程 A。由于过程 A 有形式参数 X，属于整数类型。因此，过程语句 A 必须有实际参数，如 I，J，或 -8，它们也必须属于整数类型。

在执行过程语句 A(I) 时，将实际参数 I 的数值(2)传递给形式参数 X，然后计算 X 的平方根，赋给 R，并输出 R 的值。

在执行过程语句 A(0) 时，将实际参数的值 0 传递给形式参数 X，然后计算 X 的平方根赋给 R，并输出 R 的值。

由此可知，形式参数 X 实质上是实际参数的一个“替身”。在执行过程语句以前，形式参数没有多大的实际意义，一旦执行过程语句时，它立即成为实际参数的“代办”。然而，实际

参数千变万化，“代办”只有一个。这就保证了一个过程说明可以多次调用。通常，人们把这种数值的传送称为“参数替换”。

必须指出，为了充分发挥形式参数的“代办”作用，在过程说明的执行部分，凡涉及到相应参数的地方，一律用形式参数，不能用实际参数。否则，这个过程的使用价值就很小了，甚至会出错。

例如，将过程A的执行部分改成如下形式：

```
IF I >= 0  
THEN R := SQRT(I)  
ELSE WRITELN ('SQRT OF NEGATIVE IS ILLEGAL')
```

则只有过程A(I)正确，A(J)，A(0)和A(-8)的执行结果都不对。

思考题：

如果变成如下形式，会有什么结果？

```
IF I >= 0  
THEN R := SQRT(X)  
ELSE WRITEIN ('SQRT OF NEGATIVE IS ILLEGAL')
```

此外，任何过程的形式参数的个数必须与使用的实际参数个数相等。而且要按它们在参数表中的先后次序一一对应，类型相同。

例如，过程语句A(R)，A('I')和A(0.3)都是错误的。因为R为实数类型变量，'I'是一个字符，而0.3是一个实数，都与过程说明中的形式参数X（整数类型）的类型不同。

又如，过程语句A(I, 8)、A(I, J)或A也不对。前两个语句，实际参数多于形式参数，后一个语句，实际参数少于形式参数。A没有实际参数，也就是说，它的实际参数个数为零。

如果在一个过程中有二个以上的参数，实际参数与形式参数的对应关系，是按它们出现在过程语句的实际参数表中的先后次序与出现在过程首部的形式参数表中的先后次序一一对应的。

例5-4 已知一个程序，其中有一个过程说明P，有七个过程语句。试检查其中有无语法错误。

程序如FIG PF04 PROGRAM所示：

```
PROGRAM PF04(INPUT, OUTPUT),  
VAR  
  I, J: INTEGER;  
  B: BOOLEAN;  
  CH: CHAR;  
  R: REAL;  
PROCEDURE P(X, Y: INTEGER; BN: BOOLEAN;  
            CH1: CHAR; RL: REAL);  
BEGIN  
  .  
  .  
  .  
  (• STATEMENTS •)
```

```

      .
      .
      .
END,
BEGIN  (• MAIN PROGRAM •)
  P(1, 1, B, CH, R);
  P(1, 5, B, CH, 0.5);
  P(1, 1, FALSE, 'C', R);
  P(1, J, B, CH, R, 6);
  (• TOO MANY ARGUMENTS •)
  P(1, J, B, CH);
  (• ARGUMENTS NOT ENOUGH •)
  P(8, J, B, R, CH);
  (• INCOMPATIBLE TYPE •)
  (• INCOMPATIBLE TYPE •)
  P(J, 1, B, CH, C)
  (• UNDEFINED OPERAND •)
END

```

FIG PF04 PROGRAM

在这个程序中，过程P的执行部分由一系列语句组成。本程序中未列出具体的语句。在这个程序的执行部分，七个过程语句中有三个没有语法错误，四个有语法错误。P(1, J, B, CH, R)没有语法错误是显而易见的。

P(1, 5, B, CH, 0.5)中整数5作为形式参数Y的实际参数，实数0.5作为形式参数RL的实际参数都是允许的。根据同样的替换原则，过程语句P(1, 1, FALSE, 'C', R)也是合法的。

过程语句P(8, J, B, R, CH)有两个错误。错误的性质为“矛盾的类型”(INCOMPATIBLE TYPE)。其原因在于实际参数R是实数类型，对应的形式参数CH1是字符类型；而实际参数CH是字符类型，对应的形式参数RL是实数类型。

过程语句P(J, 1, B, CH, C)的错误在于最后一个实际参数C是“未定义操作数”(UNDEFINED OPERAND)。

如果过程语句的实际参数个数多于过程说明的形式参数个数，在编译时就指出错误的性质为“参数太多”(TOO MANY ARGUMENTS)。例如过程语句P(1, J, B, CH, R, 6)。

反之，实际参数少于形式参数个数，在编译时指出错误的性质为“参数不够”(ARGUMENTS NOT ENOUGH)。例如，过程语句P(1, J, B, CH)。

此外，形式参数为实数类型时，对应的实际参数可以允许是整数类型。

§4 函数说明和函数调用

设有两个变量X和Y，如果对于X的变化范围内的每一个值，Y按一定规则有一个确定的值与之对应，则称Y是X的函数。X为自变量，Y为函数值。

一个自变量的函数叫一元函数。常用的标准函数都是一元函数。有多个自变量的函数叫多元函数。PASCAL 语言没有提供标准的多元函数，它留给用户根据需求自己去解决。

在 PASCAL 语言中，如果要使用一个非标准函数，象过程一样，同样需要做两件事：

第一，在程序的说明部分对该函数进行说明；

第二，在程序的执行部分对该函数进行调用。

函数说明与过程说明非常类似，其一般格式如图 5.4-1：

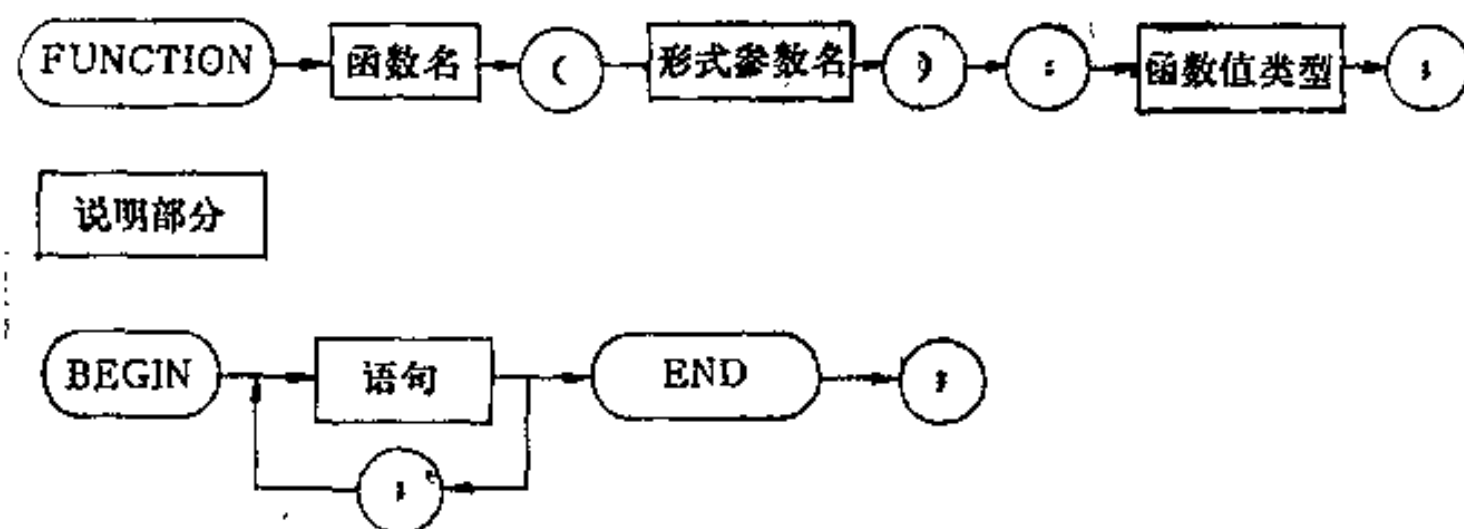


图 5.4-1 函数说明结构图

函数说明与过程说明的结构基本相同，主要有如下区别：

1. 函数说明的标志是保留关键字 FUNCTION，不是 PROCEDURE。
2. 过程说明允许没有形式参数表。但是，在一般情况下，函数说明必须有自己的形式参数，这些形式参数相当于函数的自变量。
3. 在函数说明的首部必须指出函数值的数据类型。函数的数据类型一般是标准数据类型，也可以是子界类型或指示器类型。
4. 在函数说明的执行部分至少必须有这样一个赋值语句，它将函数运算的结果 赋给函数名，从而决定函数值的大小。

函数调用与过程调用不同，过程调用是一个独立的语句，函数不能作为一个单独的语句使用，函数只能出现在表达式之中。这一点必须特别注意。

例 5-5 编制程序实现如下计算：

$$Y = X^2 + SH(X)$$

分析：

计算 X^2 比较容易，用标准函数 SQR(X) 就行了。

计算双曲正弦函数 SH(X) 没有标准函数，必须用户自己定义。

根据数学知识， $SH(X) = \frac{1}{2} (e^x - e^{-x})$

程序如 FIG PF05 PROGRAM 所示。

```

PROGRAM PF05 (INPUT, OUTPUT);
VAR
  X, Y: REAL;
FUNCTION SH (X1: REAL): REAL;
BEGIN
  SH := (EXP(X1) - EXP(-X1))/2;
END;
```

```

    BEGIN  (* MAIN PROGRAM *)
        READLN(X);
        Y:=SQR(X)+SH(X);
        WRITELN('Y=', Y)
    END
INPUT:      OUTPUT,
    2.5      Y=1.230021E+01
-7.2      Y=-6.178751E+02
15.3      Y=2.206591E+06
    FIG PF05 PROGRAM

```

例5-6 编制程序实现如下计算:

$$Y = F(U+V)/(F(U)+F(V))$$

其中: $F(T) = (A \sin^2(T) + 5)/(B \cos^2(T) - 6)$

A和B为常量, U和V为变量

分析:

计算这个问题时, 三次用函数F。但是, 函数F不是标准函数, 必须用户定义。

定义一个函数F, 其自变量为形式参数T, 实际参数为U+V, U或V。

程序如 FIG PF06 PROGRAM 所示。

```

    PROGRAM PF06 (INPUT, OUTPUT);
    CONST
        A=10; B=5;
    VAR
        U, V, Y:REAL;
    FUNCTION F (T:REAL):REAL;
    BEGIN
        F:=(A * SQR(SIN(T))+5)/(B * SQR(COS(T))-6)
    END;
    BEGIN  (* MAIN PROGRAM *)
        READLN(U, V);
        Y:=F(U+V)/(F(U)+F(V));
        WRITELN('Y=', Y);
    END.
INPUT:      OUTPUT,
    0         0      Y=5.000000E-01
-5          5      Y=9.858288E-01
10         20      Y=4.335001E-01
0.5        0.05    Y=4.028682E-01
    FIG PF06 PROGRAM

```

在这个程序中, 取常量A为10, 常量B为5。程序执行部分非常简单, 只有三条语句。即读语句, 赋值语句和写语句。

但是, 在执行赋值语句时, 它必须三次调用函数说明F, 才能计算出Y的值。为了对调用函数看得更清楚, 可以将上述一条赋值语句拆成下面五条赋值语句:

```

W:=U+V;
Y1:=F(W);
Y2:=F(U);
Y3:=F(V);
Y:=Y1/(Y2+Y3)

```

这五条语句实际上反映了程序执行的步骤，与一条赋值语句的效果相同。当然，这时必须增加一些变量说明，如 W, Y1, Y2, Y3 等。

§5 全程变量和局部变量

在过程说明和函数说明的结构中，还可以有自己的说明部分。然而，在前面的几个例题中，都没有过程或函数自己的说明部分。它们都是简单的过程或函数。

前面几个程序，全部变量都在程序的说明部分加以说明的，它们适用于整个程序，这种变量叫全程变量。反之，如果某些变量仅在某个过程之内才能使用，在过程之外则无效，这种变量叫局部变量。局部变量应该在过程的说明部分加以说明。

例5-7 编制程序计算 $Y=X^N$ ，要求用函数表示。

分析：

PASCAL 语言没有提供乘方运算的标准函数，必须由用户根据需要自己定义。

根据数学知识，当 $N < 0$ 时， $X^N = \frac{1}{X^{-N}}$

程序如 FIG PF07 PROGRAM 所示。

```

PROGRAM PF07 (INPUT, OUTPUT);

VAR
  N :INTEGER;
  X, Y:REAL;
  FUNCTION CF (X1:REAL, N1:INTEGER):REAL;
  VAR
    I:INTEGER;
    Z:REAL;
  BEGIN
    Z:=1;
    IF N1>0
    THEN FOR I:=1 TO N1 DO Z:=Z*X1,
    IF N1<0
    THEN FOR I:=1 TO -N1 DO Z:=Z/X1,
    CF:=Z;
  END;
BEGIN  (* MAIN PROGRAM *)
  X:=2;          N:=5;
  Y:=CF(X, N);   WRITELN('Y1=', Y);
  X:=-2;
  Y:=CF(X, 5);   WRITELN('Y2=', Y);

```

```

      N:=-5;
      Y:=CF(2, N);      WRITELN('Y3=', Y);
      Y:=CF(-2, -5);    WRITELN('Y4=', Y)
END.
OUTPUT,
Y1=  3.200000E+01
Y2=-3.200000E+01
Y3=  3.125000E-02
Y4=-3.125000E-02
      FIG PF07  PROGRAM

```

在这个程序中，X和N是程序的全程变量，I和Z是函数CF的局部变量。全程变量在程序的说明部分进行说明，局部变量在过程或函数的说明部分进行说明。全程变量适用于整个程序，包括过程或函数内部。局部变量仅仅适用于程序的某个过程或函数内部，离开了相应过程或函数就失效。

从物理意义上讲，全程变量在整个程序运行阶段都要占据一定的存贮单元，而局部变量仅仅在程序运行到过程语句或调用函数时才被分配一定的存贮单元。过程语句或调用函数一结束，局部变量占据的存贮单元就被释放。这就是使用局部变量的优越性——节省存贮空间。

函数CF是一个用户自己定义的二元函数。它根据乘方的幂来决定具体的计算方法。最后将计算的结果赋给函数CF。

在这个程序中，全程变量和局部变量的名称不同，区别明显。如果全程变量和局部变量的名称相同，会出现什么现象呢？

例5-8 局部变量与部分全程变量的名称相同，试分析它们运行的情况。

程序如 FIG PF08 PROGPAM 所示，

```

      PROGRAM PF08(OUTPUT),
VAR
      X, Y:INTEGER,
      PROCEDURE CHANGE,
VAR
      Y:INTEGER,
      BEGIN
        WRITELN('X2=', X:2, '    Y2=', Y:2);
        X:=2;          Y:=2;
        WRITELN('X3=', X:2, '    Y3=', Y:2);
      END,
BEGIN      (* MAIN PROGRAM *)
      X:=1;      Y:=1;
      WRITELN('X1=', X:2, '    Y1=', Y:2)
      CHANGE,
      WRITELN('X4=', X:2, '    Y4=', Y:2)
END.
OUTPUT,

```

```

X1=1  Y1=1
X2=1  Y2=0
X3=2  Y3=2
X4=2  Y4=1

```

FIG PF08 PROGRAM

在这个程序中，过程CHANGE没有形式参数表。它有局部变量Y，Y是整数类型。但是，程序有两个全程变量X和Y，其中Y和局部变量名称相同。

为了清晰地反映它们的联系及区别，主程序及过程中专门加了写语句。

观察这个程序的运行结果，人们不禁要问：

1. 为什么 $X_2 = X_1 = 1$, $X_4 = X_3 = 2$?
2. 为什么 $Y_2 \neq Y_1$, $Y_2 = 0$, $Y_4 \neq Y_3$, $Y_4 = Y_1 = 1$?

实际情况是这样的：

1. 因为X是全程变量，因此，它适用于整个程序，包括过程CHANGE之中。即在主程序与过程中用的是同一个变量X。

主程序一开始，X被赋值为1，所以 $X_1 = 1$ ；

进入过程CHANGE时，X值带进过程中，所以 $X_2 = X_1 = 1$ 。

执行过程CHANGE时，X又被赋值为2，所以 $X_3 = 2$ ；

离开过程CHANGE后，X值又带回到主程序中，所以 $X_4 = X_3 = 2$ 。

2. 变量Y与X不完全相同。它既是全程变量名，又是局部变量名。过程中用到的Y是局部变量Y，过程外用到的Y则是全程变量Y。从物理意义上讲，全程变量Y始终分配一个固定的存贮单元，而局部变量Y仅在过程执行时暂时被分配一个存贮单元。

在进入过程CHANGE之前，全程变量Y被赋值为1，所以 $Y_1 = 1$ ；

在进入过程CHANGE时，全程变量Y不参加运算，其值不变。局部变量Y尚未赋值，所以 $Y_2 = 0$, $Y_2 \neq Y_1$ ；

在执行过程CHANGE的整个期间，全程变量Y不参加运算，其值保持主程序中被赋值的结果。那里用到的Y，都是局部变量，Y被赋值为2，所以 $Y_3 = 2$ ；

在离开过程CHANGE后，分配局部变量Y的存贮单元被释放，其数值2不能传送出去。全程变量Y维持原状，所以 $Y_4 = Y_1 = 1$, $Y_4 \neq Y_3$ 。

事实上，在过程CHANGE之内与之外用到的Y，是完全不同的两个变量。

通常，人们把一个变量的有效范围，称为变量的辖域。它表示在程序的哪一部分或哪一段程序中可以使用这个变量。在一个程序中，每个变量的有效范围，必须是严格地确定的。在没有嵌套的程序中，一般有以下两条规则：

1. 全程变量适用于整个程序，局部变量适用于说明它的那个过程或函数。
2. 在局部变量与全程变量重名的情况下，相同的名称代表着不同辖域的变量，不管它们的类型是否相同。在执行过程语句时，全程变量虽然存在，但是不起作用，只有局部变量参加运算。在不执行过程语句时，只有全程变量参加运算。局部变量已不存在，更不能起作用。

事实上，不仅变量有一个辖域问题，而且标号、常量、类型、过程和函数都有一个辖域问题。局部的过程说明和函数说明就是嵌套现象，后面将要讨论。局部的类型定义与局部变量一样。

例5-9 标号和常量的辖域问题我们通过具体的例子来加以说明。
程序如 FIG PF09 PROGRAM所示:

```

PROGRAM PF09(INPUT, OUTPUT),
  LABEL
    1, 2, 3;
CONST
  X=12B; Y=6.5;
VAR
  I, J:INTEGER,
  K:CHAR,
  PROCEDURE P,
  LABEL
    2;
CONST
  V=5;
VAR
  A, B:INTEGER,
  C:CHAR,
  BEGIN
    READLN(A);
    B:=A * X * Y;
    IF B=0
    THEN GOTO 2
    ELSE
      BEGIN
        WRITELN('B<>0');
        GOTO 1
      END;
    C:=CHR(A);
  2:
    BEGIN
      WRITELN('B=0');
      WRITELN('END OF P');
      GOTO 3
    END
  END;
BEGIN      (* MAIN PROGRAM *)
  READLN(I);
  J:=TRUNC(I * X * Y);
  IF J=0
  THEN
    BEGIN
      WRITELN('J=0'); P

```



```

        END
        ELSE GOTO 2;
        K:=CHR(I);
2:
        BEGIN
            WRITELN('J<>0');
            GOTO 1
        END;
3:
1:
        WRITELN('END OF PROGRAM')
        END.
INPUT:      OUTPUT:
0           J=0
0           B=0
           END OF P
           END OF PROGRAM
0           J=0
1           B<>0
           END OF PROGRAM
1           J<>0
           END OF PROGRAM

```

FIG PF09 PROGRAM

在这个程序中，程序说明部分有标号 1, 2, 3，它们是全程标号。在程序的执行部分至少必须有三个转移语句：GOTO 1, GOTO 2 和 GOTO 3。在主程序的执行部分还必须正好有三个标号语句，

1: 语句;
 2: 语句;
 3: 语句;

它们的顺序可以颠倒。语句可以是复合语句，也可以是空语句。

在过程 P 的说明部分有标号 2，它是局部标号。在过程 P 的执行部分必须至少有一个转移语句：GOTO 2。还必须正好有一个标号语句，

2: 语句;

在过程 P 中的转移语句 GOTO 1 和 GOTO 3 是转到主程序执行部分。在过程 P 中的转移语句 GOTO 2 则转到过程执行部分，而不是转到主程序执行部分。

在主程序执行部分的转移语句 GOTO 1 和 GOTO 2 是转到主程序执行部分，GOTO 2 不能转到过程中的执行部分。

必须注意，从过程向主程序转移是合法的，从主程序向过程转移是非法的。在嵌套的情况下，从内层向外层转移是合法的，但从外层向内层转移是非法的。全程标号不能用于过程内部。

常量 X 为八进制数 12，即十进制 10。它是全程常量。常量 Y 为实数 6.5，它也是全

量。

在语句 $J := \text{TRUNC}(I * X * Y)$ 中, X 为 10, Y 为 6.5, 都是全程常量起作用。

但是, 在过程 P 中定义的常量 Y 为整数 5, 它是局部常量。在过程执行部分的语句 $B := A * X * Y$ 中, X 为 10, Y 为 5。前者是全程常量, 后者是局部常量。此时, 全程常量 Y 仍为实数 6.5, 它不参加过程 P 中的运算。

思考题:

1. 根据输入和输出的情况去分析程序是如何执行的?
2. 如果将语句 3.1 移到过程 P 中, 行不行? 为什么?

必须指出, 在程序设计过程中, 为了程序清晰可靠, 易读易懂, 希望全程变量和局部变量尽量用不同的标识符, 避免不必要的混乱。同时也希望把只在过程或函数中使用的变量说明为局部变量。

下面一个问题也是耐人寻味的思考题:

有人将例 5-7 的程序中的局部变量 I 和 Z 都改成全程变量, 程序运行的结果没有变化, 于是就得出结论:

“为了简化程序, 将局部变量都改成全程变量, 对程序运行毫无影响”。

这个观点是否正确? 为什么? 下面的例题将进一步帮助读者回答这个问题。

例 5-10 下面两个程序, 其中一个程序将变量 Z 作为局部变量, 另一个程序将变量 Z 作为全程变量。程序运行的结果不同。为什么?

程序 1 如 FIG PF10A PROGRAM 所示。

```
PROGRAM PF10A(OUTPUT);
VAR
  X, Y: INTEGER;
  PROCEDURE P;
  VAR
    Z: INTEGER;
  BEGIN
    Z := X + Y + Z;
    WRITELN('Z = ', Z:6)
  END;
BEGIN (• MAIN PROGRAM •)
  X := 2;
  Y := 3;
  P;
  P;
  P
END.
OUTPUT:
Z =    5
Z =    5
Z =    5
```

FIG PF10A PROGRAM

程序 2 如 FIG PF10B PROGRAM 所示。

```

PROGRAM PF10B(OUTPUT),
VAR
  X, Y, Z:INTEGER,
PROCEDURE P,
BEGIN
  Z:=X+Y+Z;
  WRITELN('Z=', Z:8)
END,
BEGIN (• MAIN PROGRAM •)
  X:=2;
  Y:=3;
  P;
  P;
  P
END.

```

OUTPUT:

Z= 5

Z= 10

Z= 15

FIG PF10B PROGRAM

§6 数值参数和变量参数

前面讨论的过程或函数，有的没有形式参数表，有的有一个简单的形式参数表。形式参数都是数值参数。程序设计时，还常常用到另外一种性质的形式参数——变量参数。

下面是形式参数表更常用的格式，如图5.6-1，

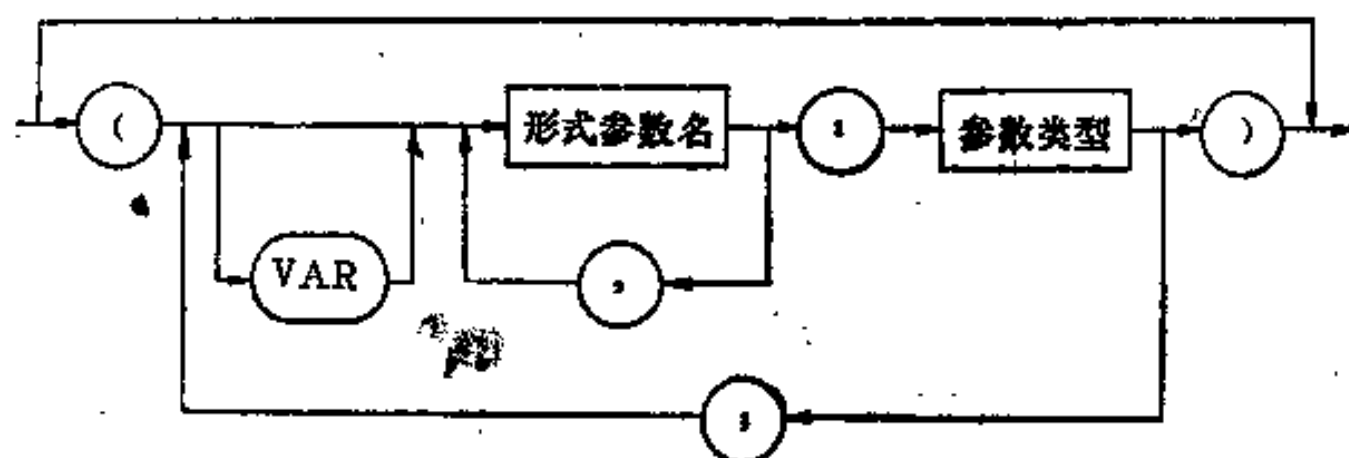


图 5.6-1 形式参数的另一种格式

从语法图上看，变量参数和数值参数是很类似的，唯一的区别在于要用关键字 VAR 作为变量参数的前缀。但是，务请注意，在过程调用时，参数的传递方式却完全不同。

例5-11A 在一个程序中，有两个非常相似的函数P和Q，但是，其中P用数值参数，Q用变量参数。试从运行结果去分析两种参数的不同特点，

程序如 FIG PF11A PROGRAM 所示。

```

PROGRAM PF11A(OUTPUT);
VAR

```

```

X, Y, M1, M2, N1, N2:INTEGER;
FUNCTION P(A:INTEGER):INTEGER;
BEGIN
    A:=A+2;
    P:=SQR(A)
END;
FUNCTION Q(VAR B:INTEGER):INTEGER;
BEGIN
    B:=B+2;
    Q:=SQR(B)
END;
BEGIN  (• MAIN PROGRAM •)
    X:=1;          Y:=1;
    WRITELN('X0=', X:2, ', Y0=', Y:2);
    M1:=P(X);      N1:=Q(Y);
    WRITELN('X1=', X:2, ', Y1=', Y:2);
    M2:=P(X);      N2:=Q(Y);
    WRITELN('X2=', X:2, ', Y2=', Y:2);
    WRITELN('M1=', M1:2, ', N1=', N1:2);
    WRITELN('M2=', M2:2, ', N2=', N2:2)
END.
OUTPUT:
X0=1, Y0= 1
X1=1, Y1= 3
X2=1, Y2= 5
M1=9, N1= 9
M2=9, N2=25
FIG PF11A PROGRAM

```

在这个程序中，函数P用数值参数，函数Q用变量参数，其余部分基本相同。由于它们的相同之点，使其运行结果有若干相同之处，由于它们的形式参数性质不同，运行的另一些结果有根本区别：

1. 在执行函数调用之前，X和Y都赋值为1。所以， $X_0=1$ ， $Y_0=1$ 。
2. 在第一次调用函数P和Q时，两个函数的自变量相同，两个函数的执行部分也基本相同。所以， $M_1=(1+2)^2=9$ ， $N_1=(1+2)^2=9$ 。
3. 在调用函数P和Q之后，数值形式参数A的变化（由1变成3）并不影响对应的实际参数X的值。因此，X仍然为1，所以 $X_1=1$ ；而变量形式参数B的变化（由1变成3）则使实际参数Y的值也由1变为3。因此， $Y_1=3$ 。
4. 第二次调用函数P和Q时，它们的自变量X和Y分别为1和3。因此，它们的函数值分别为 $(1+2)^2=9$ 以及 $(3+2)^2=25$ ，即 $M_2=9$ ， $N_2=25$ 。
5. 第二次调用函数P和Q之后，数值形式参数A又由1变为3，但仍然不影响对应的实际参数X的值。因此，X保持为1，所以 $X_2=1$ ；而变量形式参数B由3变为5，它使对应的实际参数Y的值也由3变为5。因此， $Y_2=5$ 。

思考题:

如果再增加以下语句——

$M_3 := P(X); \quad N_3 := Q(Y);$

$M_4 := P(X); \quad N_4 := Q(Y);$

则 M_3 、 M_4 、 N_3 和 N_4 分别为多少? 为什么?

上面的程序中用了两个函数, 分别采用数值形式参数和变量形式参数。实际上, 在同一个过程中也可以同时使用这两类参数。

例5-11B 在一个程序中, 有一个过程 MODIFY。它有两个形式参数 X 和 Y 。其中 X 为变量形式参数, Y 为数值形式参数。试从运行结果去分析两种参数的不同特点。

程序如 FIG PF11B PROGRAM 所示。

```
PROGRAM PF11B(OUTPUT);
VAR
  A, B: INTEGER;
PROCEDURE MODIFY(VAR X: INTEGER; Y: INTEGER);
BEGIN
  WRITELN('X1=', X:2, ', Y1=', Y:2);
  X:=X+1;      Y:=Y+3;
  WRITELN('X2=', X:2, ', Y2=', Y:2)
END;
BEGIN      (• MAIN PROGRAM •)
  A:=1;      B:=3;
  WRITELN('A1=', A:2, ', B1=', B:2);
  MODIFY(A, B);
  WRITELN('A2=', A:2, ', B2=', B:2)
END.
OUTPUT:
A1=1, B1=3
X1=1, Y1=3
X2=2, Y2=6
A2=2, B2=3
FIG PF11B PROGRAM
```

在这个程序中, 全程变量 A 和 B 是过程语句 MODIFY 的实际参数, 它们分别对应于过程 MODIFY 的变量形式参数 X 和数值形式参数 Y 。如何分析程序运行的结果呢? 只要弄清楚下面两个关键问题, 其它问题就迎刃而解了。

1. 过程语句的实际参数如何传递给过程说明的形式参数?

即为什么 $X_1 = A_1 = 1$, $Y_1 = B_1 = 3$?

2. 过程说明的形式参数的数值变化怎样影响过程语句的实际参数的值?

即为什么 $A_2 = X_2 = 2$, $B_2 \neq Y_2$, $B_2 = B_1 = 3$?

实际上, 程序的执行情况是这样的:

1. 在过程语句执行之前, 程序对全程变量即实际参数进行赋值运算, $A := 1$; $B := 3$ 。所以, $A_1 = 1$, $B_1 = 3$ 。

2. 在过程语句开始执行时, 实际参数值传递给对应的形式参数。此时, $X = A$, $Y = B$ 。所以, $X_1 = A_1 = 1$, $Y_1 = B_1 = 3$ 。

3. 在过程语句执行时, 实际参数值不变, 形式参数参加运算。 $X := X + 1$; $Y := Y + 3$ 。此时, X 变为 2, Y 变为 6。所以, $X_2 = 2$, $Y_2 = 6$ 。

4. 在过程语句执行之后, 变量形式参数传递给对应的实际参数。数值形式参数不能进行传递, 对应的实际参数值不变。此时, $A = X$, $B = Y$ 。即 $A_2 = X_2 = 2$, $B_2 = Y_2 = 6$ 。

关于变量参数及数值参数的有关问题及其主要特点归纳如下:

1. 参数形式的选择

为了传递过程运算的结果, 一般选择变量参数。为了保护实际参数不受过程运算的影响, 通常选择数值参数。

2. 参数性质的定义

定义变量形式参数时, 在参数前面以 VAR 作为前缀符号。定义数值参数时, 不需要任何前缀符号。

3. 对实际参数的要求

变量形式参数对应的实际参数只能是变量。而数值形式参数对应的实际参数允许是表达式。最简单的表达式是一个常量或一个变量。

例如, 上面程序中的过程语句 $MODIFY(A, B); MODIFY(B, A); MODIFY(A, 5); MODIFY(B, 2 * B + 5)$ 是合法的。因为变量参数 X 对应的实际参数 A 或 B 都是变量。数值参数 Y 对应的实际参数都是表达式。

但是, $MODIFY(5, B); MODIFY(2 * B + 5, A)$ 是“非法的赋值”(ILLEGAL ASSIGNMENT)。因为 X 对应的实际参数不是变量, 前者是一个常量, 后者是一个表达式。

4. 参数传递的规律

首先, 在开始执行过程语句时, 变量形式参数直接接受对应实际参数 (即对应的变量) 的地址, 而数值形式参数接受对应实际参数 (一个表达式) 的运算结果。

例如, 已知 $A = 1$, $B = 3$, 在开始执行过程语句 $MODIFY(A, 2 * B + 5)$ 时, $X = A = 1$, $Y = 2 * 3 + 5 = 11$ 。

其次, 在执行相应过程说明的执行语句的时候, 都是形式参数在参加运算。

最后, 在过程语句执行结束时, 变量形式参数的值直接传递给对应的实际参数。而数值形式参数的值不能传递给对应的实际参数, 对应的实际参数维持过程语句执行前的数值。

5. 参数传递的物理意义

参数传递是主程序 (通过过程语句) 和子程序 (即过程说明) 之间进行信息通讯的重要渠道。在 PASCAL 语言中, 变量参数的传递是通过传送地址来实现的, 数值参数的传递是通过传送数值来实现的。

传送地址是指把实际参数的地址传送给对应的形式参数。在过程中每个变量形式参数都有一个相应的单元, 称为形式单元。形式单元用来存放对应的实际参数的地址。(注意, 不是实际参数的数值)。当程序执行过程语句时, 首先, 将实际参数的地址送给对应的形式单元。然后, 过程体对形式参数的任何引用或赋值都通过对形式单元的间接访问 (间址) 来实现。也就是说, 在过程执行期间, 对用到的每个变量形式参数的访问 (读或赋值), 都是通过间址操作而在相应的实际参数上进行的。

传送数值是一种简单的参数传递方法。首先，将实际参数（表达式）的数值进行计算并将结果存放到对应的形式单元。

然后，对形式单元的内容进行运算，就象对局部变量进行运算一样。这时，形式单元中的数值不再与实际参数单元的数值发生关系。实际参数单元维持原来的数值。

为了加深理解，现在再概括为以下三点：

〈1〉开始时，变量参数的形式单元中存放的是实际参数单元的地址，数值参数的形式单元中存放的是实际参数单元的数值，或其计算的结果。

〈2〉在运算时，变量参数是对形式单元进行间址操作，数值参数是对形式单元进行直接操作。

〈3〉在结束时，变量参数的实际单元中存放的是间址操作的结果。数值参数的实际单元中存放的是原来的数值。

在过程或函数说明时，函数名本身是这个函数的形式参数。所有形式参数都是属于某个过程或函数的局部变量；并且不必重新加以说明。在调用过程或函数时，这些局部变量在存储器中都要分配形式单元。函数名相当于调用函数时可以访问的一个变量，这是它与过程的一个重要区别。

在程序设计时，形式参数和实际参数尽量用不同的标识符；执行部分都用形式参数，不用实际参数。这两点对数值参数特别重要。

例5-11C 有一个过程 MODIFY，它共有三个数值形式参数 X、Y 和 Z。在这个过程的执行部分既用了形式参数，又用了实际参数。试从运行的结果去体会过程执行部分使用形式参数的重要性。

程序如 FIG PF11C PROGRAM 所示。

```
PROGRAM PF11C(OUTPUT);
VAR
  A, B, C:INTEGER;
PROCEDURE MODIFY(X, Y, Z:INTEGER);
BEGIN
  X:=X+3; Y:=B+3; C:=C+3;
  WRITELN('X=', X:2, ', Y=', Y:2, ', Z=', Z:2)
END;
BEGIN (• MAIN PROGRAM •)
  A:=3; B:=3; C:=3;
  WRITE('1:'); MODIFY(A, B, C);
  A:=3; B:=3; C:=3;
  WRITE('2:'); MODIFY(5, 5, 5);
  A:=3; B:=3; C:=3;
  WRITE('3:'); MODIFY(2*A+5, 2*B+5, 2*C+5)
END.
```

OUTPUT:

```
1: X= 6, Y=6, Z= 3
2: X= 8, Y=6, Z= 5
3: X=14, Y=6, Z=11
```

FIG PF11C PROGRAM

在这个程序中，实际参数A、B和C对应于形式参数X、Y和Z。初看起来，它们运算的结果似乎应该相同，实际情况却是另一回事。

其中：

语句 $X := X + 3$ ；这是形式参数参加运算，也是通常使用的方法。

在情况1时 X初值为 $A = 3$ ，运算后X变为6。即 $3 + 3 = 6$ 。

在情况2时 X初值为5。运算后X变为8。即 $5 + 3 = 8$ 。

在情况3时 X初值为11，即 $2 * 3 + 5 = 11$ 。运算后X变为14。即 $11 + 3 = 14$ 。

语句 $Y := B + 3$ ；这是形式参数和实际参数混合使用。由于B总是为3，所以Y总是6。形式参数没有起到传递作用。

语句 $Z := C + 3$ ；这是实际参数进行运算，与形式参数无关。形式参数Z等于形式参数的初值或形式参数表达式的运算结果。所以，Z分别为3、5和11。

此外，如果参数类型不是标准的数据类型，而是用户定义的其它类型，比如数组类型。在程序设计时要注意以下两点：

1. 参数类型只能用类型标识符。

为了保证实际参数与形式参数的一致性，在形式参数的类型不是标准数据类型时，必须在主程序或它的外层过程中定义这种类型，然后才能将类型标识符（类型名）用到形式参数表中去。

2. 参数通常被定义为变量参数。

如果参数是一个容量很大的构造类型数据，如一个很大的数组，用数值参数时，在过程调用时，必须为它分配一个能够存放整个数组的存储区（等于实际参数单元数），而用变量参数时，只需要一个存放数组起始地址的形式单元。这样做，不仅大大节省了过程调用期间所占据的内存空间，而且节省了参数替换时复制整个数组值的时间。其代价是降低了访问每个数组元素的速度，即每次访问多了一次间址操作。

这两点将在第七章中进一步讨论。

• §7 嵌套和递归

过程（包括函数——下同）的嵌套是实现结构式程序设计的主要手段。嵌套和递归则是PASCAL语言程序设计的一个重要组成部分。本节主要介绍嵌套、递归及向前引用等基本概念。

什么叫嵌套？

嵌套就是一层套一层的程序结构。比如，过程A套住过程B，过程B套住过程C，过程C套住过程D……。嵌套中要强调的是必须完全套住，局部套住是不行的，即不允许有两个或两个以上的过程彼此交叉，或称相互“跨骑”的现象。

例如，图5.7-1表示的程序是非法的。

这个图中，过程A和B相互套住一部分，但是，谁也不能完全套住另一方。这是错误的，无法运行的。

例5-12 从一个“过程——函数——过程”的嵌套来分析嵌套的规则。

程序如 FIG PF12A PROGRAM 所示。

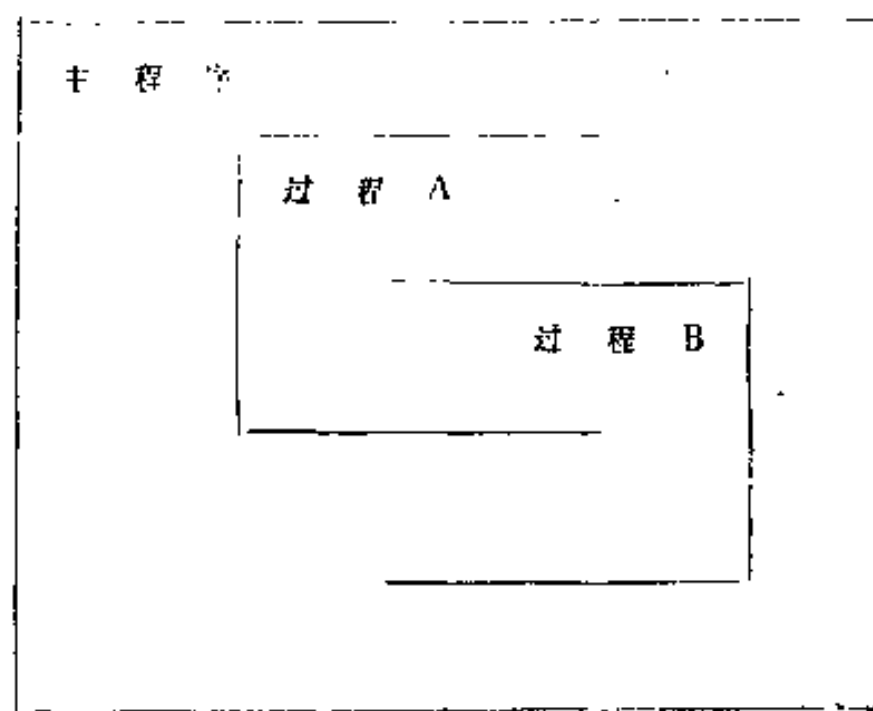


图 5.7—1 嵌套中的跨骑

```

PROGRAM PF12A (OUTPUT),
VAR
  I, J: INTEGER,
  PROCEDURE P1,
    FUNCTION F (I2:INTEGER):INTEGER,
    PROCEDURE P2 (VAR I1:INTEGER);
    BEGIN
      I1:=2*I;  WRITELN ('I1=', I1:4)
    END;
  BEGIN
    P2 (I2);  F:=I2*3
  END;
  BEGIN
    J:=F (I)
  END;
BEGIN          (* MAIN PROGRAM *)
  I:=10;
  WRITELN ('I =', I:4);
  P1;
  WRITELN ('J =', J:4)
END.
OUTPUT:
I =10
I1=20
J =60
FIG PF12A  PROGRAM

```

在这个程序的说明部分，过程 P1、P2 和函数 F 都没有自己的局部变量。过程 P1 没有形式参数，函数 F 有数值参数，过程 P2 有变量参数。它们之间的关系非常密切，这个关系就是嵌套关系。过程 P1 套住函数 F，函数 F 套住过程 P2。

在这个程序的执行部分，主程序调用过程 P 1。在过程 P 1 的执行部分，过程 P 1 调用函数 F。在函数 F 的执行部分，函数 F 调用过程 P 2。它们调用的基本规则是上一层调用下一层。

这种嵌套关系可以用图 5.7-2 来表示。

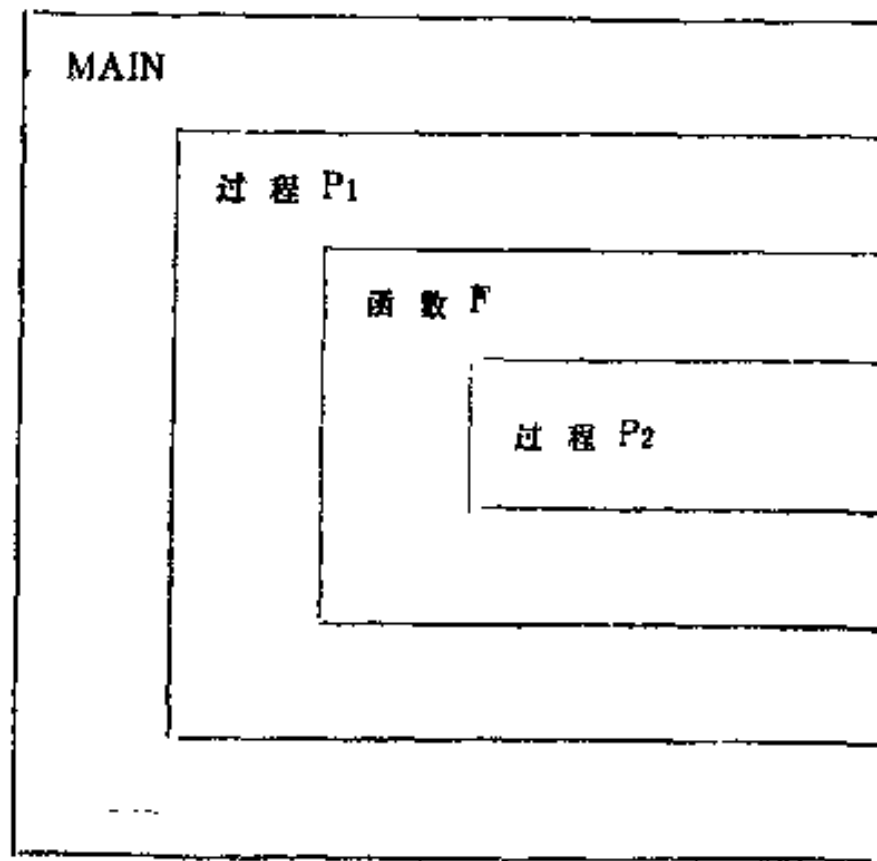


图 5.7-2 嵌套关系示意图

从图上可以看出，它们之间是逐层相包、完全套住的。套住其它过程的过程叫外层过程。被其它过程套住的过程叫内层过程。对于多层嵌套来说，内层和外层是相对的。为了准确地表达嵌套的层次，通常将嵌套按从外向内的顺序进行编号，并把相应过程的层号称为过程的嵌套深度。有的将主程序编为 1 层（也有编为 0 层的），然后从外向内依次为 2 层，3 层，4 层，……。在这里，主程序为 1 层，过程 P 1 为 2 层，函数 F 为 3 层，过程 P 2 为 4 层。

由上述所讲，1 层可以调用 2 层，2 层可以调用 3 层。是否可以隔层调用呢？

FIG PF12B PROGRAM 可以回答这个问题。

```

PROGRAM PF12B (OUTPUT);
VAR
  I, J: INTEGER;
PROCEDURE P1;
  FUNCTION F (I2: INTEGER): INTEGER;
    PROCEDURE P2 (VAR I1: INTEGER);
      BEGIN
        I1 := 2 * I1;
        WRITELN ('I1 =', I1: 6)
      END;
    BEGIN
      P2(I2);
      P := I2 * 3
    END;
  BEGIN
    P2(1);
  
```

```

                (• UNDEFINED SYMBOL •)
                J := F (I)
            END;
BEGIN          (• MAIN PROGRAM •)
    I := 10;
    WRITELN (' I =', I : 6);
    P 1;
    WRITELN (' J =', J : 6);
    J := F (I);
                (• UNDEFINED OPERAND •)
                (• MISSING OPERATOR •)
    P 2 (J)
                (• UNDEFINED SYMBOL •)
END.

```

FIG PF12B PROGRAM

回答是否定的，隔层不能调用。

主程序只能调用主过程 P 1，不能调用函数 F 和过程 P 2；过程 P 1 只能调用函数 F，不能调用过程 P 2。如果违背了这一规则，在编译时，则把 P 2 作为“未定义的符号”(UNDEFINED SYMBOL)；把 F 作为“未定义的操作数”(UNDEFINED OPERAND)，并认为 F (I) 中“遗漏了运算符”(MISSING OPERATOR)。

前面肯定了外层对内层调用的合理性，否定了外向内隔层调用的可能性。那么，反过来，内层可以对外层调用吗？甚至内层可以隔层调用外层吗？

FIG PF12C PROGRAM 能够回答这个问题。

```

PROGRAM PF12C (OUTPUT);
VAR
    I, J: INTEGER;
    PROCEDURE P1;
        FUNCTION F (I2: INTEGER): INTEGER;
            PROCEDURE P2 (VAR I1: INTEGER);
                BEGIN
                    I1 := 2 * I1;
                    J := F(2);
                    P1
                END;
            BEGIN
                P2 (I2);
                F := I2 * 3;
                P1
            END;
        BEGIN
            J := F (I);
        END;
    BEGIN          (• MAIN PROGRAM •)

```

```

I:=10;
WRITELN (' I =', I:6);
P1;
WRITELN (' J =', J:6)
END.
NOTE:
RUNNING TIME IS TOO LONG;
RUNNING RESULT IS
'RESERVED INSTRUCTION TRAP'.
FIG PF12C PROGRAM

```

回答是肯定的。不仅内层可以对外一层调用，而且可以隔层调用。就嵌套深度来表示，可以用图5.7-3表示。

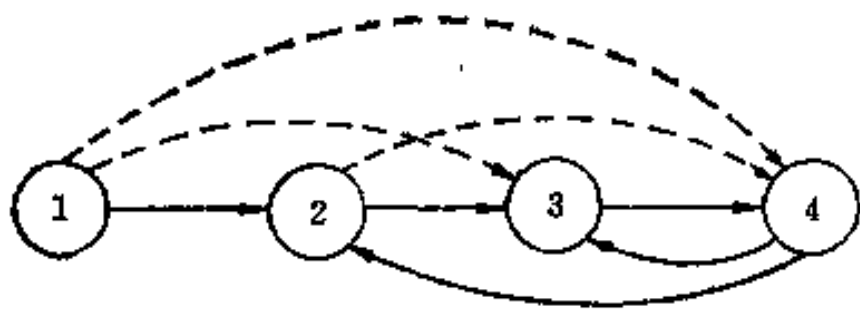


图 5.7-3 嵌套调用示意图

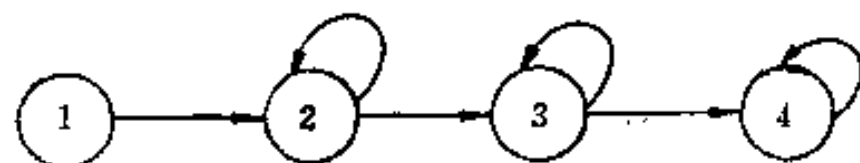


图 5.7-4 嵌套深度及调用示意图

其中：实线表示允许调用，

虚线表示禁止调用。

此外，任何过程都不能调用主程序。这一点在示意图中省去了。

既然下层可以调上一层，上二层，……，只要不调主程序。那么，过程可以调自己吗？

回答是肯定的。

这时的嵌套深度示意图见图5.7-4。

程序如FIG PF12D PROGRAM所示。

```

PROGRAM PF12D (OUTPUT);
VAR
  I, J: INTEGER;
PROCEDURE P1;
  FUNCTION F (I2:INTEGER):INTEGER;
    PROCEDURE P2 (VAR I1: INTEGER);
      BEGIN
        I1:=2 * I1;
        P2 (I1)
      END;
    BEGIN
      P2(I2);
      F:=3 * I2 * F (I2)
    END;
  BEGIN
    J:=F (I);

```

```

      P1
    END;
  BEGIN  (• MAIN PROGRAM •)
    I:=10;
    WRITELN ('I =', I:6);
    P1;
    WRITELN ('J =', J:6)
  END.
NOTE:
  RUNNING TIME IS TOO LONG;
  RUNNING RESULT IS
  'RESERVED INSTRUCTION TRAP'.
  FIG PF12D PROGRAM

```

这种过程调用自己以及下层过程调用上层过程的调用规则，常称为过程的递归调用规则。其中过程调用自己的递归叫直接递归，内层过程调用外层过程的递归叫间接递归。

递归调用在完成阶乘运算、级数运算，幂指函数运算等方面特别有效。但是，不能不分场合地随意递归调用，有时会降低运算速度，甚至招致某些错误的结果。

上面两个程序 (PF12C 和 PF12D) 虽然没有语法错误，但是无法正常运行。它们运行的时间太长，原因就在于它们不断递归调用，没有结束递归的条件。它们运行的最终结果是进入“保留指令陷阱” (RESERVED INSTRUCTION TRAP)。原因在于每递归调用一次，都要分配存贮区，最终存贮区被分配完毕，程序被迫停止运行。凡是有使用价值的程序，递归调用总是有条件的。

例 5-13 编制程序计算组合 C_m^n 。比如 C_5^3 。

分析：

计算组合时，有两个常用公式。

$$C_m^n = C_m^{m-n}$$

$$C_m^n = C_{m-1}^{n-1} + C_{m-1}^n$$

前一个公式常用于 $m < 2n$ 时，后一个公式是实际计算组合，不断递归执行。递归调用结束的条件是： $C_m^0 = 1$ 和 $C_m^m = 1$ 。

程序如 FIG PF13 PROGRAM 所示。

```

PROGRAM PF13 (INPUT, OUTPUT);
VAR
  M, N, Z, I: INTEGER;
FUNCTION ZH (X, Y: INTEGER): INTEGER;
  (• C (M, N)=C (M, M-N) •)
  (• C (M, N)=C (M-1, N-1) +C (M-1, N) •)
BEGIN
  IF X < 2*Y
  THEN Y:=X-Y;
  I:=I+1;
  IF Y=0
  THEN ZH:=1

```

```

        ELSE
        IF    Y=1
        THEN ZH:=X
        ELSE ZH:=ZH (X-1, Y-1) +ZH (X-1, Y)
    END;
BEGIN    (• MAIN PROGRAM •)
    M:=8;
    N:=5;
    Z:=ZH (M N);
    WRITELN ('I=', I:3, ' ', Z=' ', Z:4)
END.
OUTPUT:
I= 29, Z= 56

```

FIG PF13 PROGRAM

在这个程序中，函数 ZH 的执行部分两次调用自己，进行直接递归调用。递归调用的基本步骤如下： $C_i = C_i + C_i$ ($C_i = C_i + C_i$)

$C_i = C_i + C_i$ ($C_i = C_i + C_i$)

$C_i = C_i + C_i$

$= 6 + C_i + C_i$

$= 6 + 5 + C_i + C_i$

$= 6 + 5 + 4 + C_i + C_i$

$= 6 + 5 + 4 + 3 + C_i$ ($C_i = C_i + C_i$)

$= 6 + 5 + 4 + 3 + 3$

$= 21$

同理 $C_i = C_i + C_i$

$C_i = 15$

$C_i = 20$

所以 $C_i = 21 + 15 + 20 = 56$

变量 I 用于计算递归调用的次数， C_i 需要递归调用 29 次，结果为 56。

思考题：

C_i 等于多少？要递归调用多少次？是否与 C_i 情况相同？

上面介绍的嵌套和递归只有纵向关系，没有横向关系。在同一层中只有一个过程。如果在同一层中有一个以上过程，则程序设计时就要复杂一些。

例 5-14 在一个程序中有两个并列的过程 P1 和 P2。先说明 P1，后说明 P2。试分析它们的相互调用关系。

程序如 FIG PF14A PROGRAM 所示。

```

PROGRAM PF14A (INPUT, OUTPUT),
VAR
    A, B: INTEGER;
    PROCEDURE P1 (Y: INTEGER);
    BEGIN

```

```

        Y:=2*Y+10;
        B:=Y DIV 30
    END;
PROCEDURE P2 (VAR X: INTEGER);
BEGIN
    IF X<400
    THEN P1 (X)
    END;
BEGIN (• MAIN PROGRAM •)
    READLN (A) ;
    P2 (A);
    WRITELN ('B=', B:4)
END.

```

INPUT:	OUTPUT:
21	B= 1
152	B= 10
504	B= 504
-505	B= -33
-156	B= -10
-28	B= -1

FIG PF14A PROGRAM

在这个程序中，P1和P2的嵌套深度都为2，属于同一层。过程P2在一定条件下调用过程P1，这是允许的。

既然后说明的过程可以调用先说明的过程。那么，反过来是否可以呢？这需要增加一点新内容，否则是不行的。在编译时，会将过程P1中调用的P2作为“未定义的符号”（UNDEFINED SYMBOL）。

程序如FIG PF14B PROGRAM所示。

```

PROGRAM PF14B (INPUT, OUTPUT);
VAR
    A, B : INTEGER;
PROCEDURE P1 ( Y:INTEGER);
BEGIN
    Y:=2*Y+10;
    B:=Y DIV 30;
    P2
    (• UNDEFINED SYMBOL •)
END;
PROCEDURE P2 (VAR X:INTEGER);
BEGIN
    IF X<400
    THEN P1 (X)
    END;
BEGIN (• MAIN PROGRAM •)

```

```

    READLN(A);
    P2(A);
    WRITELN ('B=', B:4)
END.

```

FIG PF14B PROGRAM

如果实际工作中只需要过程 P1 调用过程 P2, 那么只要在设计程序时先说明过程 P2, 后说明过程 P1。

如果不仅需要过程 P1 调用过程 P2, 而且需要过程 P2 调用过程 P1, 在 PASCAL 语言程序设计时, 就要加上“向前引用”的说明。

程序如 FIG PF14C PROGRAM 所示。

```

PROGRAM PF14C (INPUT, OUTPUT);
VAR
    A, B, N:INTEGER;
    PROCEDURE P2 (VAR X:INTEGER);
    FORWARD;
    PROCEDURE P1 (Y:INTEGER);
    BEGIN
        Y:=2 * Y + 10;
        B:=Y DIV 30;
        P2 (Y)
    END;
    PROCEDURE P2;
    BEGIN
        IF X<400
        THEN
            BEGIN
                N:=N+1;
                P1(X)
            END
        END;
    BEGIN (• MAIN PROGRAM •)
        READLN (A);
        P2 (A);
        WRITELN ('N=', N:3, ' , B=', B:4)
    END.

```

INPUT:	OUTPUT:
21	N= 4 B= 16
152	N= 2, B= 21
504	N= 0, B= 504
-505	N= 7, B= 72
-158	N= 8, B= 938
-28	N= 11, B= 956

FIG PF14C PROGRAM

这个程序所反映的调用规则也是间接递归调用规则。变量N表示的是递归调用的次数。在输入的变量A小于400时开始递归调用。这就是递归调用的条件。否则，不需要递归调用。此时， $N=0$ 。

两个并列过程都要间接递归调用时，必须增加“向前引用”(FORWARD)的说明。所谓向前引用说明，以本程序为例，就是对原来的程序作如下修改：

1. 在过程P1前面加上过程P2的首部和关键字FORWARD。如：

```
PROCEDURE P2 (VAR X:INTEGER);  
FORWARD;
```

2. 取消过程P2说明中的形式参数表。变成PROCEDURE P2。

如果有三个并列的过程P1，P2和P3。其中P1调用P2，P2调用P3，P3调用P1。主程序调用过程P1。

程序设计时，可以有两种办法：

第一种方法是将过程P2和过程P3“向前引用”。三个过程说明的顺序是P1，P2和P3。程序加FIG PF14D PROGRAM所示。其中变量A表示递归调用的结果，变量N表示递归的次数。在A小于100的情况下开始递归调用。

```
PROGRAM PF14D (INPUT, OUTPUT),  
VAR  
  A, N:INTEGER;  
  PROCEDURE P3 (Z:INTEGER);  
  FORWARD;  
  PROCEDURE P2 (Y:INTEGER);  
  FORWARD;  
  PROCEDURE P1 (VAR X:INTEGER),  
  BEGIN  
    IF X<100  
    THEN  
      BEGIN  
        N:=N+1;  
        X:=X+10;  
        P2 (X)  
      END  
    END;  
  PROCEDURE P2,  
  BEGIN  
    P3 (Y)  
  END;  
  PROCEDURE P3,  
  BEGIN  
    P1 (Z)  
  END;  
BEGIN (* MAIN PROGRAM *)  
  READLN (A);
```

```

P1 (A),
WRITELN ('N=', N:3, ' , A=' , A:3)
END.

```

INPUT :	OUTPUT:
1	N= 10, A= 11 (A=101)
10	N= 9, A= 20 (A=100)
55	N= 5, A= 65 (A=105)
60	N= 4, A= 70 (A=100)
90	N= 1, A=100 (A=100)
120	N= 0, A=120 (A=120)

NOTE:

IF ABOVE FORMAL PARAMETER ALL ARE VARIABLE
PARAMETER, THEN RESULT IS (. . . .).

FIG PF14D PROGRAM

第二种方法是将过程 P1 “向前引用”，三个过程说明的顺序是 P3，P2 和 P1，程序如 FIG PF 14 E PROGRAM 所示，显然，第二种方法比第一种方法简单，这在程序设计时是一种技巧。

思考题：

1. 经过多次递归调用后，为什么输出的 A 仅仅比输入的 A 大 10？
2. 如果将过程 P2 和 P3 用到的参数都改成变量参数，输入输出的关系会有变化吗？为什么？

提示：如果所有参数都是变量参数，则输出结果不小于 100，见括号内的内容 (……)。

```

PROGRAM PF14E (INPUT, OUTPUT),
VAR
  A, N: INTEGER,
PROCEDURE P1 (VAR X: INTEGER),
FORWARD,
PROCEDURE P3 (VAR Z: INTEGER),
BEGIN
  P1 (Z)
END,
PROCEDURE P2 (VAR Y: INTEGER),
BEGIN
  P3 (Y)
END,
PROCEDURE P1,
BEGIN
  IF X < 100
  THEN
    BEGIN
      N := N + 1,
      X := X + 10,
      P2 (X)
    END
  END
END

```

```

        END
    END,
    BEGIN  (• MAIN PROGRAM •)
        READLN (A);
        P1 (A);
        WRITELN ('N=', N:3, ' , A=' , A:4)
    END.

```

INPUT :	OUTPUT:
1	N= 10, A= 101
10	N= 9, A= 100
55	N= 5, A= 105
60	N= 4, A= 100
90	N= 1, A= 100
120	N= 0, A= 120

FIG PF14E PROGRAM

前面讨论的过程有一个特点。它们或者只有纵向的关系，或者只有横向的关系，无论纵向或横向的关系，都可以嵌套或递归。

下面将程序进一步扩充，变成“纵横交错”的关系，这时仍然可以嵌套或递归。

例5-15 有一个程序，它有两个并列的过程P1和P2，过程P1又有两个并列的小过程P1A和P1B，过程P2也有两个并列的小过程P2A和P2B。试分析它们之间的关系。

程序如 FIG PF15A PROGRAM 所示。

```

PROGRAM PF15A (OUTPUT);
PROCEDURE P1;
    PROCEDURE P1A;
        BEGIN
            WRITELN ('PROCEDURE P1A1 ');
        END;
    PROCEDURE P1B;
        BEGIN
            WRITELN ('PROCEDURE P1B1 ');
            P1A
        END;
    BEGIN
        WRITELN ('PROCEDURE P11 ');
        P1A;
        P1B
    END;
PROCEDURE P2;
    PROCEDURE P2A;
        BEGIN
            WRITELN ('PROCEDURE P2A1 ');
        END;
    PROCEDURE P2B

```

```

        BEGIN
            WRITELN ('PROCEDURE P2B1 ',),
            P2A
        END,
    BEGIN
        WRITELN ('PROCEDURE P21 ',),
        P1,
        P2A,
        P2B
    END,
    BEGIN (• MAIN PROGRAM •)
        P1,
        P2,
    END.
OUTPUT:
PROCEDURE P1,
PROCEDURE P1A,
PROCEDURE P1B,
PROCEDURE P1A,
PROCEDURE P2,
PROCEDURE P1,
PROCEDURE P1A,
PROCEDURE P1B,
PROCEDURE P1A,
PROCEDURE P2A,
PROCEDURE P2B,
PROCEDURE P2A,

```

FIG PF15A PROGRAM

选用这个程序，是要讨论嵌套关系。从嵌套深度看，主程序的嵌套深度为 1。过程 P 1 和过程 P 2 的嵌套深度都为 2，它们是并列关系。过程 P1A、过程 P1B、过程 P2A 和过程 P2B 的嵌套深度都为 3。这种嵌套关系可以用图 5.7-5 来表示。

这个关系框图形象地表示，2 层的 P 1 和 P 2 是并列关系，3 层的 P1A 和 P1B 之间、P2A 和 P2B 之间也是并列关系。在这种情况下，根据前面的知识，主程序可以调用 P 1 和 P 2，P 2 可以调用 P 1、P2A 和 P2B。P 1 可以调用 P1A 和 P1B。同时，P1B 可以调用 P1A，P2B 可以调用 P2A。这些调用表现为两条规则：任何过程都可以调用直接隶属于它的下一层过程；在一个过程中的后说明的过程可以调用先说明的过程。

如果违背了这些基本规则，则会在编译时指出程序设计的错误。

程序如 FIG PF15B PROGRAM 所示。

```

PROGRAM PF15B (OUTPUT),
PROCEDURE P1,
    PROCEDURE P1A,
    BEGIN

```

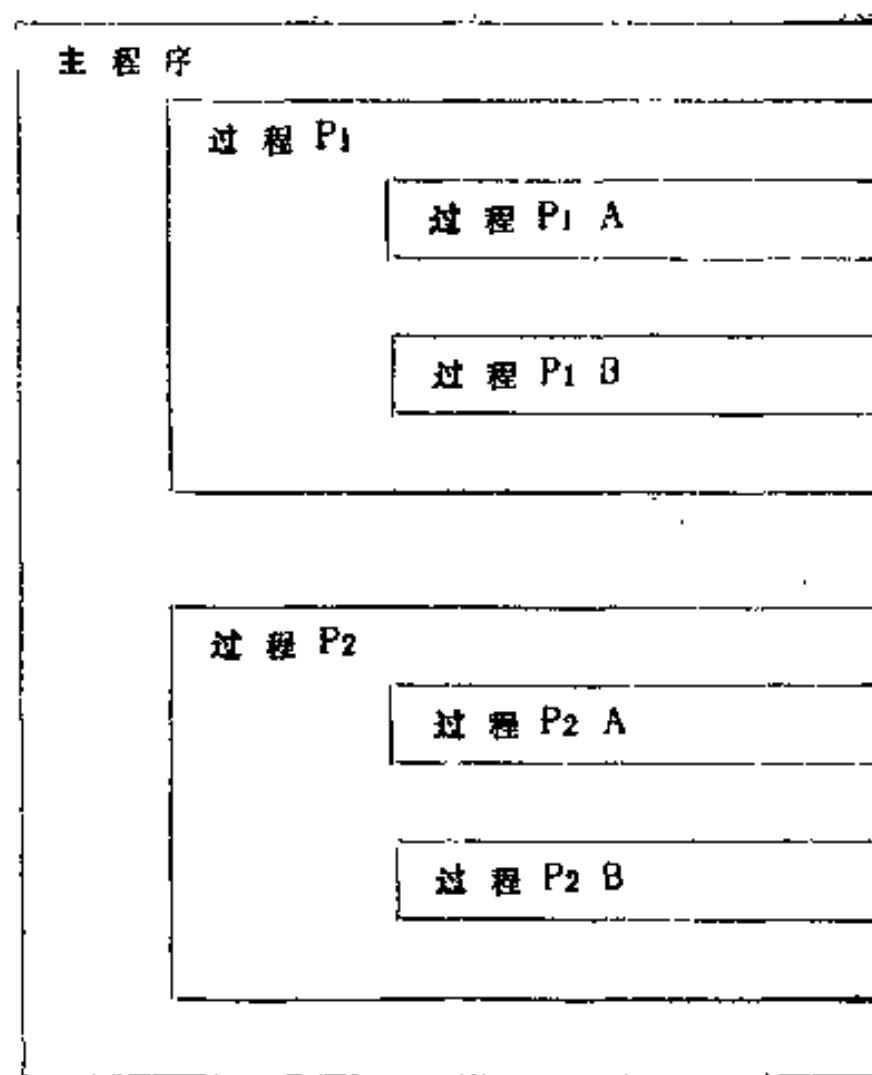


图 5.7—5 另一种嵌套关系示意图

```

WRITELN ('OUTPUT P1A, ');
P1B  (• UNDEFINED SYMBOL •)
END,
PROCEDURE P1B,
BEGIN
  WRITELN ('OUTPUT P1B, ')
END,
BEGIN
  WRITELN ('OUTPUT P1, ');
  P2,      (• UNDEFINED SYMBOL •)
  P1A,
  P2B      (• UNDEFINED SYMBOL •)
END,
PROCEDURE P2,
  PROCEDURE P2A,
  BEGIN
    WRITELN ('OUTPUT P2A, ');
    P2B,    (• UNDEFINED SYMBOL •)
    P1A,    (• UNDEFINED SYMBOL •)
  END,
  PROCEDURE P2B,
  BEGIN
    WRITELN ('OUTPUT P2B, ');
    P1B     (• UNDEFINED SYMBOL •)
  END,

```

```

BEGIN
    WRITELN ('OUTPUT P2, ');
    P2A,
    P1B    (• UNDEFINED SYMBOL •)
END,
BEGIN    (• MAIN PROGRAM •)
    P1,
    P2,
    P1A,    (• UNDEFINED SYMBOL •)
    P2B    (• UNDEFINED SYMBOL •)
END.

```

— FIG PF15B PROGRAM

在这个程序中有许多错误，其错误性质大致分为三类：

1. 违背了直接隶属关系。

例如主程序调用 P1A 和 P2B，是外层隔层调用内层，又如过程 P2 调用 P1B，过程 P1 调用 P2B，也不是正确的外层对内层的调用。

2. 违背了先说明、后调用的规则。

例如过程 P1 调用过程 P2，过程 P1A 调用过程 P1B，过程 P2A 调用过程 P2B 都是错误的。只有后说明的过程才可以调用先说明的过程。

3. 违背了属于“同一个过程”的条件。

如果是不属于同一过程的两个过程，即使某一个过程已经在先说明，另一个过程也不能

调用它。如过程 P2A 不能调用过程 P1A，过程 P2B 不能调用过程 P1B。

这种复杂的嵌套关系可以用图 5.7-6 示意图来表示。

图中表示的是能够实现的调用关系，对于不能实现的关系一律省去了。

关于这种复杂程序的递归问题，包括直接递归和间接递归问题。其基本思想与前面介绍的相同，本文不再复述。

综合上述嵌套与递归的全部讨论，可以归纳出如下几点相互调用的规则：

1. 主程序只能调用 2 层过程，只

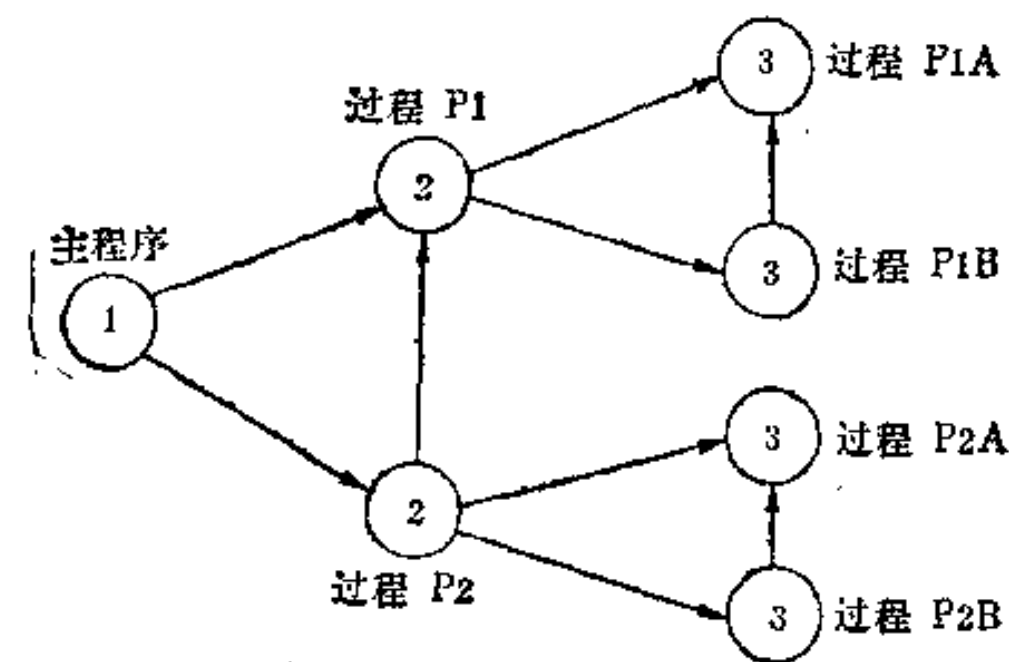


图 5.7-6 嵌套允许调用示意图

有 2 层过程直接隶属于主程序。任何过程不能调用主程序。

2. 每个过程只能调用直接隶属于它的那一层过程。其嵌套深度比主程序调用过程多 1 级。不是直接隶属关系的不能调用。

3. 每个过程都能调用与自己同隶属于某一过程的另一个过程。两者之间是同层关系，嵌套深度相同。两者又都属于同一个过程。

同层过程相互调用时，一般应遵循先说明、后调用的原则。即后说明的过程可以调用先说明的过程。在特殊情况下，采用“向前引用”说明的方法，先说明的过程也可以调用后说明的过程。这种方法主要用于间接递归调用。

4. 每个过程都可以调用自己。这是同层调用的特殊情况。通常把这种调用称为直接递归调用。

5. 每个过程还可以调用那些嵌套着自己的过程。包括直接或间接嵌套着自己的过程（不包括主程序）。通常把这种调用也称为直接递归调用。

在程序设计过程中，无论直接递归调用，还是间接递归调用，一般都是在一定条件下的递归调用。否则，就会出现无穷次递归调用的问题。它常常浪费计算机的大量运行时间，甚至耗费它的整个存储空间，也无法得到预期的结果。这在程序设计时是不允许出现的。

无穷次递归是行不通的。无限次嵌套也是不允许的。各种版本的 PASCAL 语言都有自己的规定。OMSI PASCAL 语言最多允许嵌套十层。这也是程序设计中要注意的问题。

由于嵌套的关系，程序设计中的标号、常量、变量、类型、过程和函数的辖域问题就非常突出，在 GOTO 语句的使用上特别突出。在“全程变量和局部变量”一节中已经说明了一些问题，下面再举例作进一步的说明。

例5-16 转移语句在过程中的应用。

一个过程中的转移语句可以转到过程内的任何地方，也可以转到主程序中。

一个内层过程中的转移语句可以转到内层过程内的其它地方，也可以转到嵌套着自己的外层过程中去。

主程序中的标号只能用于主程序的执行部分，过程中的局部标号只能用于相应过程的执行部分。标号说明了不使用是不行的。

程序如 FIG PF16A PROGRAM 所示。

```
PROGRAM PF16A(INPUT, OUTPUT);
LABEL
  20;
VAR
  X, Y:REAL;
  PROCEDURE IMPOUT;
  LABEL
    10;
  BEGIN
    Y:=X-1;
    IF Y=0
    THEN GOTO 10
    ELSE
      BEGIN
        X:=X*(X+1);
        WRITELN('X=', X);
        GOTO 20;
      END;
  10 :
    WRITELN ('V=0.0')
  END;
BEGIN          (• MAIN PROGRAM •)
```

```

        READLN(X);
        IF      X>=0
        THEN    JMPOUT
        ELSE    WRITELN('X<0');
20:
    END.
INPUT:      OUTPUT:
-8          X<0
1           Y=0.0
8           X=7.200000E+01

```

FIG PF16A PROGRAM

反过来，如果转移语句不是同层内转移或向外层转移，而是由主程序或外层向过程或内层过程转移，则是非法的。

程序如 FIG PF16B PROGRAM 所示。

```

        PROGRAM PF16B(INPUT, OUTPUT);
LABEL
    20:
VAR
    X, Y:REAL;
    PROCEDURE JMPIN;
    LABEL
        10:
        BEGIN
            Y:=X-1;
            IF      Y<>0
            THEN    GOTO 10
            ELSE    GOTO 20;
            X:=X*(X+1);
        20:
            (• LABEL DEFINED AT WRONG LEVEL •)
            WRITELN('Y=0')
        END;
            (• MISSING LABEL DEFINITION •)
    BEGIN
        READLN(X);
        IF      X>=0
        THEN    JMPIN
        ELSE    WRITELN('X<0');
    10:
        (• LABEL NOT DECLARED •)
    END.
    (• MISSING LABEL DEFINITION •)
FIG PF16B PROGRAM

```


在这个程序中，全程标号20在主程序的执行部分没有使用。局部标号10在过程的执行部分没有使用。因此，在编译时都认为它们遗漏了标号定义“(MISSING LABEL DEFINITION)”而标号语句10；出现在主程序中执行部分，编译时认为它是“未说明的标号”(LABEL NOT DECLARED)。标号语句20；出现在过程的执行部分；编译时认为它“标号定义在错误的层次”(LABEL DEFINED AT WRONG LEVEL)。由此可知，标号说明是非常严格的。在程序设计时要特别注意。

在“全程变量和局部变量”一节中，讨论过在没有嵌套的程序中处理变量辖域的两条规则。现在，我们更加完整地阐述如下：

1. 全程变量适用于整个程序，局部变量适用于说明它的那个过程。

在嵌套的情况下，外层过程的局部变量适用于整个外层过程，包括它所嵌套的内层过程。内层过程的局部变量只适用于内层过程本身。

2. 在变量重名的情况下，在执行过程语句时，全程变量虽然存在，但是不参加运算，只有局部变量参加运算。在不执行过程语句时，只有全程变量参加运算，此时，局部变量实际上不存在（因为没有被分配存储单元），更不会参加运算。

在嵌套的情况下，执行内层过程的执行语句时，只有内层局部变量参加运算。全程变量和外层局部变量虽然存在，但是不参加运算。在不执行内层过程的执行语句但执行外层过程的执行语句时，外层过程的局部变量参加运算。全程变量虽然存在，但是不参加运算，内层局部变量实际上不存在，更不会参加运算。

综合上面两条规则，我们可以得到变量名的优先顺序规则：

在一个内层过程的执行语句中出现一个变量名时，首先判断这个变量是否为该内层过程的局部变量。如果是，则作为自己的局部变量使用。

如果不是，继续判断这个变量是否为外层过程的局部变量。如果是，则使用这个变量。

如果不是，继续向外层查找。……

如果在2层过程中也没有找到这个变量，可以到主程序中查找。如果在主程序中已经说明，则作为全程变量使用。

如果在主程序中没有说明，则作为程序错误处理。通常指出错误的性质为“未定义的符号”(UNDEFINED SYMBOL)。

如果在N层的执行语句中有变量X，则可以通过图5.7-7流程图去查找：

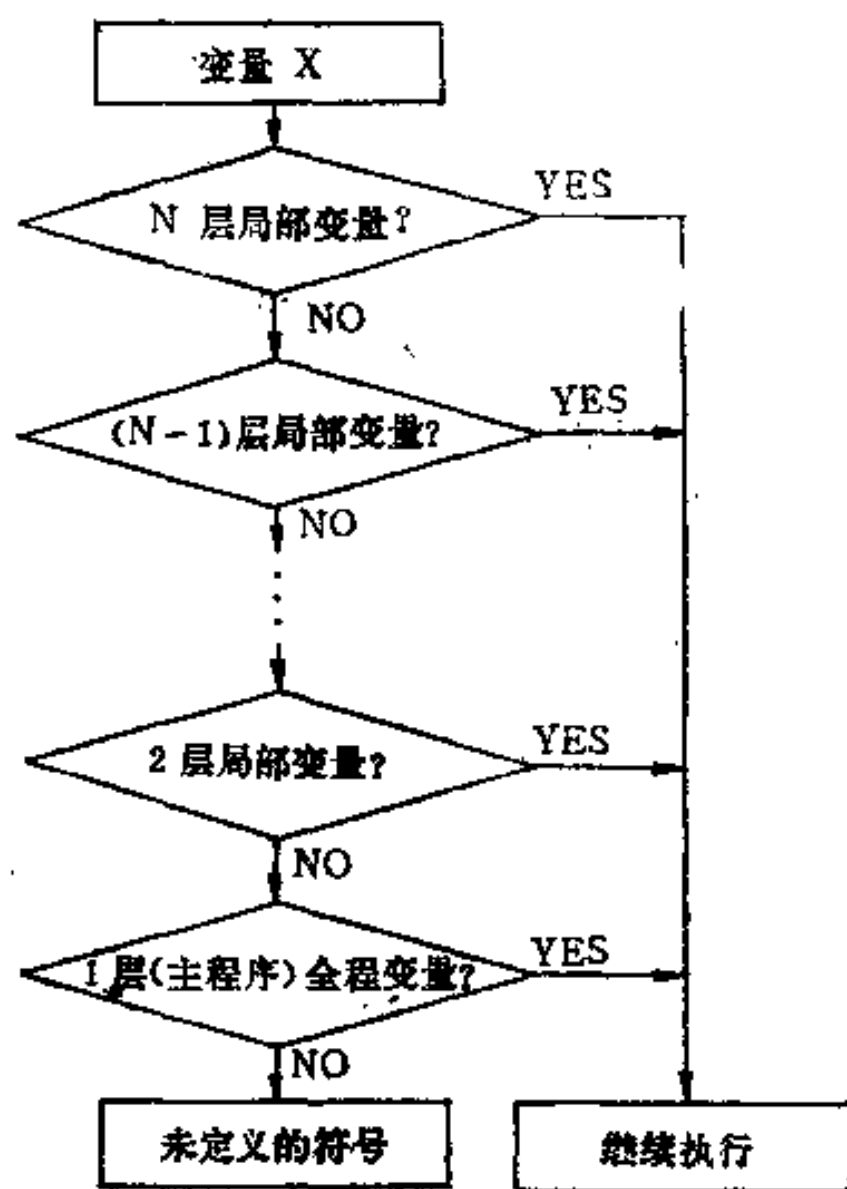


图 5.7-7 嵌套中变量查找示意图

• §8 过程参数和函数参数

PASCAL 语言不仅可以用数值参数和变量参数，而且可以用过程参数和函数参数。前

两种参数比较简单一些，是经常使用的。后两种参数相对复杂一些，应用得少一点。

图5.8-1是形式参数表完整的格式：

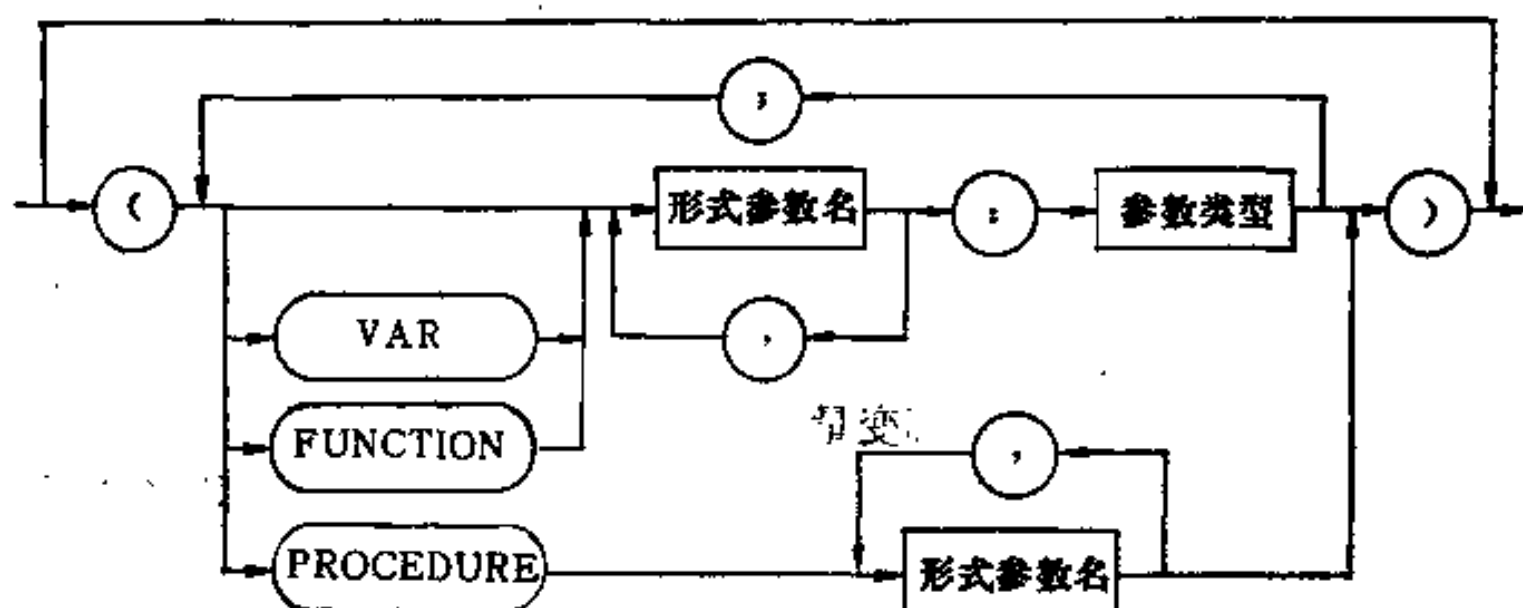


图 5.8-1 形式参数表的完整格式

前面已经讨论了数值参数和变量参数，下面主要介绍过程参数和函数参数。

一、过程参数 过程参数的实际参数为已经定义的过程名。其形式参数以关键字 PROCEDURE 作为前缀，后面跟形式参数名。

例5.17 有一个函数如下：

$$y = \begin{cases} x^2 - 2x + 2, & x \geq 50 \\ (x + 50)^2 - 2(x + 50) + 2, & x < 50 \end{cases}$$

自变量X分别为100和20，试通过带有过程参数的函数来计算函数值。

分析：

遇到这种问题，人们往往想通过变量参数的传递来解决自变量的变化问题。常见的程序如FIG PF17A PROGRAM 所示。

```

PROGRAM PF17A(INPUT, OUTPUT),
VAR
  X, Y1, Y2:INTEGER,
  PROCEDURE P(VAR X1:INTEGER),
  BEGIN
    X1 := X1 + 50
  END,
  FUNCTION F(PROCEDURE P0,
              VAR X0:INTEGER):INTEGER,
  BEGIN
    IF X0 < 50
    THEN P0(X0);
    F := SQR(X0) - 2 * X0 + 2
  END,
BEGIN      (• MAIN PROGRAM •)
  X := 100;
  Y1 := F(P, X);
  WRITELN('Y1 = ', Y1:6);
  X := 20;

```

```

        Y2:=F(P,X);
        WRITELN('Y2=', Y2:6)
    END.
OUTPUT:

```

```

Y1=    9802
Y2=    362

```

FIG PF17A PROGRAM

从程序运行的结果可以看出， $X=100$ 时，函数值 9802 是正确的。但是， $X=20$ 时，函数值 362 是错误的，应该 4762。

为什么会出现这个错误呢？初学的同志常以为没有执行过程 P，实际上过程 P 确实执行了，FIG PF17B PROGRAM 的运行结果就说明了这一点。但是，执行过程 P 时，变量参数 X1 的值是错误的。所以程序运行结果仍是错误的。

```

PROGRAM PF17B(INPUT, OUTPUT);
VAR
    X, Y1, Y2:INTEGER;
    PROCEDURE P(VAR X1:INTEGER);
    BEGIN
        WRITELN('X1=', X1:6);
        X1:=X1+50;
        WRITELN('X1=', X1:6)
    END;
    FUNCTION F(PROCEDURE P0;
                VAR X0:INTEGER):INTEGER;
    BEGIN
        WRITELN('X0=', X0:6);
        IF X0<50
            THEN P0(X0);
        WRITELN('X0=', X0:6);
        F:=SQR(X0)-2*X0+2
    END;
BEGIN      (• MAIN PROGRAM •)
    X:=100;
    Y1:=F(P, X);
    WRITELN('Y1=', Y1:6);
    X:=20;
    Y2:=F(P, X);
    WRITELN('Y2=', Y2:6)
END.

```

OUTPUT:

```

X0=    100
X0=    100
Y1=    9802
X0=     20

```

```

X1 = -11956
X1 = -11906
X0 =      20
Y2 =     362

```

FIG PF17B PROGRAM

原因在哪里?

PASCAL 语言规定, 不是所有的过程和函数都可以作为参数。作为参数的那些过程或函数的参数必须是数值参数, 不能是其它参数。

例如, 在程序说明部分定义了下面五个过程:

```

PROCEDURE P (PROCEDURE Q);
PROCEDURE A (PROCEDURE PA);
PROCEDURE B (FUNCTION F, REAL);
PROCEDURE C (VAR X, Y, REAL);
PROCEDURE D (M, N, INTEGER);

```

在程序执行部分有四个过程调用语句:

```
P(A); P(B); P(C); P(D);
```

在编译时, 虽然未必指出语法错误, 但是, 在运行时, 只有P(D)能正确执行, 因为作为过程P的参数是过程D, 而过程D的参数是数值参数。其余三个语句都不能正确执行, 因为作为参数的过程A, B, C的参数分别是过程参数, 函数参数和变量参数。当然, 作为参数的过程可以没有参数。

对于本题, 如果希望仍然用过程参数, 可以把过程P改成数值参数, 然后通过全程变量X2来传递数据。同时, 适当改变程序的局部结构。

新程序如 FIG PF17C PROGRAM 所示。

```

PROGRAM PF17C(INPUT, OUTPUT),
VAR
  X, X2, Y1, Y2:INTEGER;
PROCEDURE P(X1:INTEGER);
  BEGIN
    WRITELN('X1=', X1:6);
    X1:=X1+50;
    WRITELN('X1=', X1:6);
    X2:=X1
  END;
FUNCTION F(PROCEDURE P0;
  VAR X0:INTEGER):INTEGER;
  BEGIN
    WRITELN('X0=', X0:6);
    IF X0<50
    THEN
      BEGIN
        P0(X0); X0:=X2
      END;

```

```

        WRITELN('X0=', X0:6);
        F:=SQR(X0)-2*X0+2
    END;
BEGIN      (• MAIN PROGRAM •)
    X:=100;
    Y1:=F(P, X);
    WRITELN('Y1=', Y1:6);
    X:=20;
    Y2:=F(P, X);
    WRITELN('Y2=', Y2:6)
END.
OUTPUT:
X0=   100
X0=   100
Y1=  9802
X0=   20
X1=   20
X1=   70
X0=   70
Y2=  4762
      FIG PF17C PROGRAM

```

思考题:

如果将例5-17的三个程序中的 $P_0(X_0)$ 都改成 $P(X_0)$, 试回答:

1. 三个修改后的程序的运行结果是否相同?
2. P 和 P_0 的意义是否相同? 过程和过程参数的意义有什么区别?
3. 新程序中过程参数是否起了作用? 去掉它有没有问题?
4. 在新程序中, 过程 P 是否可以用变量参数?
5. 为什么强调过程执行部分用形式参数而不用实际参数? 这对参数传递有什么影响?

二、函数参数 函数参数的实际参数为一个函数名, 这个函数名必须是已经定义的函数。

函数参数的形式参数由三个部分组成, 它们是关键字 **FUNCTION**, 形式参数名和参数类型。一般希望形式参数和实际参数属于同一种类型, 但是, 实际参数为整数类型时, 允许形式参数为实数类型。函数参数的参数类型就是函数值的类型。

例如, 在高等数学里, 经常计算函数 $f(x)$ 的积分 $P = \int_a^b f(x)dx$ 。通过 PASCAL 语言来解决这些问题是比较简单的, 只要说明一个计算积分的过程 JF 。它可以根据不同的函数 $f(x)$ 和不同的区间 $[a, b]$ 来计算积分。

关于函数参数的格式, 我们举例说明用图5.8-2表示。

其中:

形式参数 F 为函数参数, 它对应于函数 $f(x)$ 的函数。

形式参数 A 和 B 为数值参数, 它对应于积分的区域——下界和上界。

形式参数 P 为变量参数, 它对应于积分的结果。由于积分的结果一般需要输出或参加进

PROCEDURE JF(FUNCTION F:REAL;
A,B:REAL; VAR P:REAL);

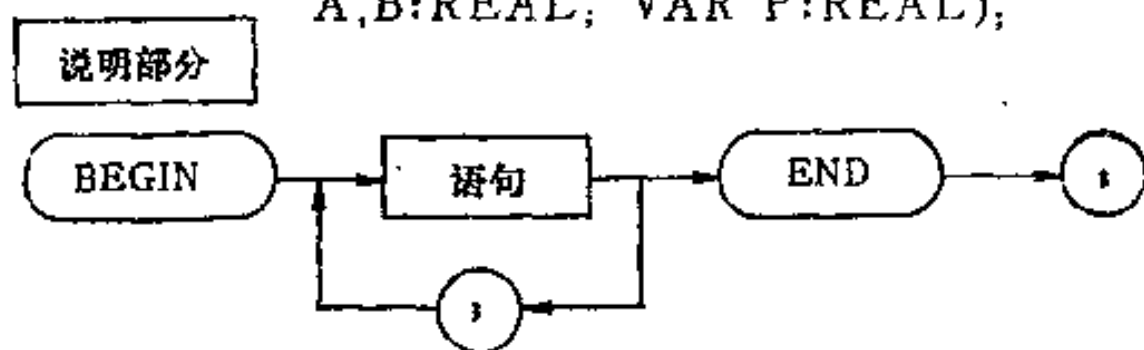


图 5.8-2 函数参数的格式举例

一步的运算，所以常用变量参数。

当然，这里的过程 JF 是一般形式，程序设计时可以酌情增减某些参数。

例 5-18 A 编制一个程序，

计算函数 $f(x)$ ，其中 $f(x) = x^2 - 2x + 2$ 。

要求：

1. 计算函数值部分用一个过程；
2. 第一次计算的函数值作为第二次计算时函数的自变量，再求函数值。

程序如 FIG PF18A PROGRAM 所示。

```
PROGRAM PF18A(OUTPUT);
VAR
  X:REAL;
FUNCTION F(Y:REAL):REAL;
BEGIN
  F:=SQR(Y)-2*Y+2
END;
PROCEDURE P(VAR Z:REAL; FUNCTION F0:REAL);
BEGIN
  Z:=F0(Z)
END;
BEGIN      (• MAIN PROGRAM •)
  X:=5;
  WRITELN('X0=', X:7:2);
  P(X, F);
  WRITELN('X1=', X:7:2);
  P(X, F);
  WRITELN('X2=', X:7:2)
END.
```

OUTPUT:

X0= 5.00

X1= 17.00

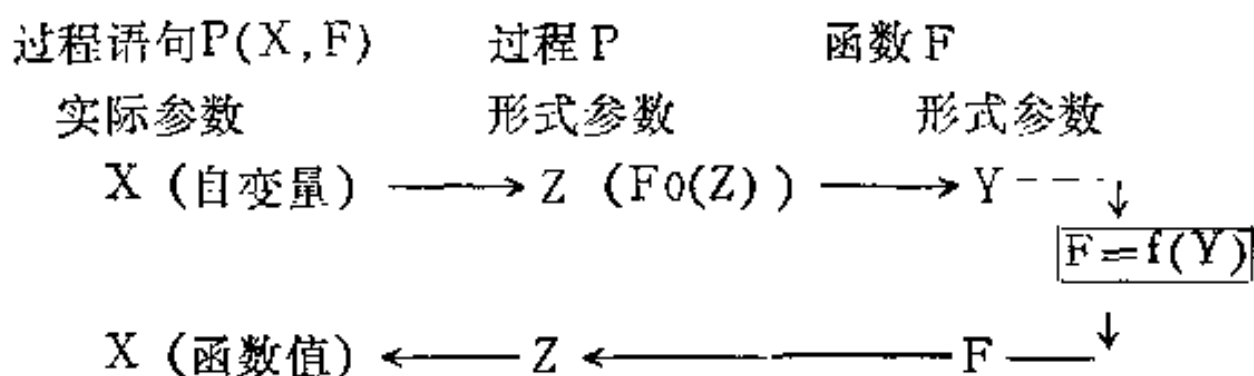
X2= 257.00

FIG PF18A PROGRAM

在这个程序中，过程 P 的作用就是将函数计算的结果赋给一个变量。自变量和函数值都是变量参数 Z。函数的计算是由函数 F 完成的。

全程变量 X 是过程 P 的实际参数，它对应于过程 P 的形式参数 Z，Z 又是函数 F（函数参数为 F0）的实际参数，它对应于函数 F 的形式参数 Y。这样的渠道就沟通了参数 X、Y 和 Z 之间的关系。

它们之间的关系如下图所示：



此外，在这个程序中，过程 P 的参数之一是函数参数 F0，而作为参数的函数 F 的参数是数值参数 Y，这就保证了参数传递的正确性。

前面的程序中，作为参数的过程和函数都是用户自己定义的，那么能否用标准过程和标准函数呢？

标准 PASCAL 语言允许直接使用。OMSI PASCAL 语言不允许这样做。在编译时，把它作为“参数有错”(BAD ARGUMENT) 处理。

解决的办法很简单，只要加一个辅助函数说明。例如，标准函数 SIN，可以用辅助函数 SINE。

```
FUNCTION SINE (X:REAL):REAL;
BEGIN      SINE:=SIN(x)      END;
```

例5-18B 编制程序计算下列函数，已知函数 $F = \sin X \cdot \cos X + \sqrt{1+x^2}$ 。

其中 X 分别为 1.5, 3.7 和 -3.7。

要求程序用函数参数。

程序如 FIG PF18B PROGRAM 所示

```
PROGRAM PF18B(OUTPUT);
VAR
  X:REAL;
  FUNCTION SINE(Y:REAL):REAL;
  BEGIN
    SINE:=SIN(Y)
  END;
  FUNCTION COSINE(Y:REAL):REAL;
  BEGIN
    COSINE:=COS(Y)
  END;
  FUNCTION F(FUNCTION S, C:REAL;
             VAR Z:REAL):REAL;
  BEGIN
    F:=S(Z) * C(Z) + SQR(1+SQR(Z))
  END;
BEGIN      (• MAIN PROGRAM •)
  X:=1.5;
  WRITELN('F1=', F(SINE, COSINE, X));
  X:=3.7;
  WRITELN('F2=', F(SINE, COSINE, X));
```

```

      X:=-3.7;
      WRITELN('F3=', F(SINE, COSINE, X));
END.
OUTPUT:
F1=1.873336E+00
F2=4.282108E+00
F3=3.383399E+00
      FIG    PF18B    PROGRAM

```

这个程序的运行表明，不仅过程中可以用函数参数，函数中可以用过程参数。实际上，无论过程说明还是函数说明，都允许用一个或几个函数参数或过程参数，甚至两者都用。

由于作为参数的过程或函数本身只能用数值参数或不用参数，这就影响了参数传递的能力。因此，过程参数用得较少。

思考题：

如果去掉主程序中的三个赋值语句，而在写语句中将 X 改成对应的数值。这样行不行？为什么？

* §9. 外部过程和外部函数

前面几节介绍的所有过程和函数，都是内部过程和内部函数。它们都是由程序员在主程序内部加以具体说明的，并且具有以下两个重要特点：

第一，必须把它们与主程序同时加以编译，产生统一的一个目标文件；

第二，它们与主程序之间以及它们相互之间的调用关系，在编译之后已经完全解决，不需要再经过链接程序另行处理。

从前面的例题中可以看到，内部过程（函数）的说明与调用是比较方便的。但是，为了方便更多的用户，提高系统效率，实现文件共享等目的，PASCAL 语言编译程序，允许使用外部过程和外部函数。这些外部过程和外部函数，都是由程序员在主程序外部加以具体说明，即独立于主程序之外，单独作为一个文件存放和处理的。它们与内部过程（或函数）类似，但又有区别。一般地说，它们具有以下两个重要特点：

1. 必须把它们与主程序分别加以编译，产生各自的目标文件；

2. 它们与主程序之间以及它们相互之间的调用关系，在编译之后，必须通过链接程序专门处理。

既然内部过程和内部函数的说明和应用已经比较方便，为什么还要用外部过程和外部函数呢？这主要由于以下四条理由：

1. 可以把许多用户经常使用的过程和函数，做成一个程序库。

例如，许多用户经常使用各种数学函数、文件管理及输入输出处理过程，可以把它们作为程序库。如 OMSI PASCAL 语言中的 PASCAL.OBJ 就是一个程序库，它里面存放的就是标准过程和标准函数。

运用程序库可以缩短编译程序，提高编译过程中内存的使用效率。同时，每个用户在使用这些过程时，不必加以说明，从而提高了用户的工作效率。

2. 可以把某些用户专用的过程或函数作为外部过程或外部函数。

这些专用的过程或函数在加工完成之后，可以加到标准程序库中去，并对编译程序作相应的轻微改动。也可以将这些过程或函数单独作为另一个库文件，以供链接程序专门处理。

3. 可以把待调试的过程或经常需要修改的过程作为外部过程处理。

在设计与调试一些大中型程序时，往往采用分段调试或分块调试，而不必每次都把那些调试好的部分与待调试的部分（如一个过程）一起进行完整编译。这时可以把待调试的过程作为一个外部过程，单独编译，校正语法错误，最后通过链接程序与调试好的主程序链接。根据同样的道理，把那些经常需要修改的过程作为外部过程。这样做，可以节约时间，降低调试成本，提高系统的使用效率。

4. 当链接（装配）程序具有复盖处理功能时，可以对使用的外部过程进行复盖处理，以减少该程序进行过程中占用的内存单元，这在运行一个大程序出现内存容量不够时，是特别有效的一项措施。

如何编制外部过程呢？下面通过一个具体的例题来讨论内部过程和外部过程的联系及区别。

例5-19 分别用内部过程的方法和外部过程的方法编制程序，求出1000以内的亲密数对 (THE PAIRS OF AMICABLE NUMBER)。

分析：

什么叫亲密数对？

如果正整数A的因子和为B，B的因子和又为A，则称A和B为一对亲密数。

正整数A的因子，指的是能够整除A的所有正整数，但是，不包括A本身。如12的因子为1，2，3，4和6。

12的因子和为 $1 + 2 + 3 + 4 + 6 = 16$ 。而16的因子和为 $1 + 2 + 4 + 8 = 15$ 。所以12和16不是一对亲密数。

6的因子和为 $1 + 2 + 3 = 6$ ，即A的因子和为A本身，这说明6和6是一对亲密数。

程序：

用内部过程的方法的程序如 FIG PF19A PROGRAM 所示。

```
PROGRAM PF19A(INPUT,OUTPUT),
VAR
  A,B,C:INTEGER;
  PROCEDURE FACTORSUM(X:INTEGER;
                      VAR Y:INTEGER);
  VAR
    I:INTEGER;
  BEGIN
    Y:=1;
    FOR I:=2 TO X DIV 2 DO
      IF X MOD I=0
        THEN Y:=Y+I
    END;
  BEGIN  (• MAIN PROGRAM •)
    FOR A:= 2 TO 1000 DO
      BEGIN
```

```

        FACTORSUM(A,B);
        FACTORSUM(B,C);
        IF      (A=C) AND(A<>B)
            THEN WRITELN(A:6,B:6)
        END
    END.
OUTPUT,
220      284
284      220

```

FIG PF19A PROGRAM

用这种方法编制的程序只有一个文件 (PF19A.PAS)。

程序执行部分是一个循环语句，求出1000以内的亲密数对。

循环体部分先求出A的因子和B，再求出B的因子和C，最后根据A，B，C的情况决定是否输出。只有在 $A=C$ ，并且 $A \neq B$ 时才输出亲密数对A和B。这样输出的亲密数对排除了 $A=B$ 的情况。如6和6。

计算因子和是通过过程 FACTORSUM 来实现的。其中赋值语句用于初始化因子和，循环语句用于累计因子和。由于整数X的最大因子不大于X的一半。所以，循环次数最多为A的一半，且取整数。循环体为一个条件语句，如果控制变量I是该整数X的一个因子，则将I加入到因子和中去。

用外部过程的方法的程序如 FIG PF19B PROGRAM 所示，它是由三个文件组成的。

```

(* FILE1.....GLOBAL.PAS *)
(* ..... *)
PROGRAM PF19B(INPUT,OUTPUT);
VAR
    A,B,C:INTEGER;
(* FILE2.....EXTNAL.PAS *)
(* ..... *)
PROCEDURE FACTORSUM(X:INTEGER;
                    VAR Y:INTEGER);
VAR
    I:INTEGER;
BEGIN
    Y:=1;
    FOR I:=2 TO X DIV 2 DO
        IF X MOD I=0
            THEN Y:=Y+I
    END;
(* FILE3.....MAIN.PAS *)
(* ..... *)
PROCEDURE FACTORSUM(X:INTEGER;
                    VAR Y:INTEGER);
EXTERNAL;
BEGIN      (* MAIN PROGRAM *)

```

```

FOR A:=2 TO 1000 DO
  BEGIN
    FACTORSUM(A,B);
    FACTORSUM(B,C);
    IF (A=C) AND (A<>B)
      THEN WRITELN(A:6,B:6)
  END
END.
OUTPUT,
220      284
284      220

```

FIG PF19B PROGRAM

这个程序与前面的程序基本相同，主要区别在于将内部过程方法的一个源程序文件“一分为三”，形成三个独立的文件：

1. 全程变量文件：GLOBAL.PAS

它的内容基本上是程序的首部及程序的说明部分。唯一的区别在于减去了独立出去的过程说明部分。

2. 外部过程文件：EXTNAL.PAS

它的内容就是独立出去的过程 FACTORSUM。其实质是内部过程独立化。

3. 主程序文件：MAIN.PAS

它的内容基本上是程序的执行部分。唯一的区别在于在执行部分之前增加了过程 FACTORSUM 的首部和关键字 EXTERNAL。关键字 EXTERNAL 表示用到的过程 FACTORSUM 是一个外部过程。

如何运行这个程序呢？

其基本思想是首先在全程变量文件配合下，分别将外部过程文件和主程序文件进行编译，然后分别进行汇编，最后将它们汇编的结果（目的程序）和 PASCAL 库程序进行链接并运行。

对于这一个具体程序，根据 OMSI PASCAL 语言的规定，它的运行步骤如下：

```

. R PASCAL
  * MAIN=GLOBAL,MAIN
. R PASCAL
  * EXTNAL=GLOBAL,EXTNAL/E
. R MACRO
  * MAIN=MAIN
  * EXTNAL=EXTNAL
  * ^C
. R LINK
  * MAIN=MAIN,EXTNAL,PASCAL
  * ^C
. RUN MAIN

```

其中：GLOBAL 应放在 MAIN 和 EXTNAL 前面，不能反过来。

/E表示编译外部过程文件。如果在外部过程文件EXTNAL.PAS的开头号上{\$E+},则在运行时不必用开关/E。

带有外部过程的程序的运行,在第十章第二节有进一步的讨论。请读者提前阅看。外部函数的基本思想与外部过程完全相同,下面通过一个例题简单地说明一下。

例5-20 通过外部函数的方法计算下列函数。 $Y = SH(X) + X^N + X^2$

其中X为实数,N为整数。

分析:

其中计算 X^2 用标准函数SQR。计算双曲正弦函数SH(X)用自定义函数SH,如例5-5所示。计算乘方函数 X^N 用自定义函数CF,如例5-7所示。

程序如图PF20A PROGRAM所示,它是由四个文件组成的。

```
(• FILE1.....GLOBAL.PAS •)
(• ..... •)
PROGRAM PF20A(INPUT,OUTPUT);
VAR
  N:INTEGER;
  X,Y:REAL;
  (• FILE2.....SH.PAS •)
(• ..... •)
FUNCTION SH(X1:REAL):REAL;
BEGIN
  SH:=(EXP(X1)-EXP(-X1))/2
END;
  (• FILE3.....CF.PAS •)
(• ..... •)
FUNCTION CF(X1:REAL;N1:INTEGER):REAL;
VAR
  I:INTEGER;
  Z:REAL;
BEGIN
  Z:=1
  IF N1>0
  THEN FOR I:=1 TO N1 DO Z:=Z*X1,
  IF N1<0
  THEN FOR I:=1 TO -N1 DO Z:=Z/X1;
  CF:=Z;
END;
  (• FILE4.....MAIN.PAS •)
(• ..... •)
FUNCTION SH(X1:REAL):REAL;
EXTERNAL;
FUNCTION CF(X1:REAL;N1:INTEGER):REAL;
EXTERNAL;
BEGIN (• MAIN PROGRAM •)
```

```

      READLN(X,N);
      Y:=SQR(X)+SH(X)+CF(X,N);
      WRITELN('Y=',Y)
END.
INPUT,      OUTPUT,
2      5      Y=3.962686E+01
4      5      Y=1.067290E+03
4      -5     Y=4.329090E+01
3      -70    Y=1.901787E+01

```

FIG PF20A PROGRAM

这个程序有两个外部函数文件，一共四个文件。这四个文件的运行步骤如下，

```

. R PASCAL
* MAIN=GLOBAL,MAIN
. R PASCAL
* SH=GLOBAL,SH/E
. R PASCAL
* CF=GLOBAL,CF/E
. R MACRO
* MAIN=MAIN
* SH=SH
* CF=CF
* ^ C
. R LINK
* MAIN=MAIN,SH,CF,PASCAL
* ^ C
. RUN MAIN

```

如果有超过四个以上的外部过程或外部函数，则不能简单地模仿上述步骤进行，其程序运行步骤见第十章第二节的有关部分。

必须指出，并不要求把所有的过程和函数都作为外部过程文件或外部函数文件。只要求把必要的过程或函数独立出来，其余的过程和函数仍然留在全程变量文件中。

全程变量文件不仅可以包含程序首部和全程变量，还可以包括全程的标号定义、类型定义、内部过程和内部函数说明。全程变量文件服务于外部过程文件和主程序文件，因为编译外部过程与主程序时都会用到它，它必须单独作为一个文件。

外部过程的方法，必须至少要有三个文件。但是，内部过程的方法，也可以有几个文件。

例如，例5-20的问题，也可以用四个文件，采取内部函数的方法来运行。这时的四个文件，完全是一个完整的源程序文件的“一分为四”，不必增加函数首部及关键字 EXTERNAL。

```

( * FILE1.....GLOBAL.PAS * )
( * ..... * )
PROGRAM PF20B(INPUT,OUTPUT);
VAR

```

```

N:INTEGER;
X,Y:REAL;
(• FILE2.....SH PAS •)
(• ..... •)
FUNCTION SH(X1:REAL):REAL;
BEGIN
    SH:=(EXP(X1)-EXP(-X1))/2;
END;
(• FILE3.....CF.PAS •)
(• ..... •)
FUNCTION CF(X1:REAL;N1:INTEGER):REAL;
VAR
    I:INTEGER;
    Z:REAL;
BEGIN
    Z:=1;
    IF N1>0
    THEN FOR I:=1 TO N1 DO Z:=Z*X1;
    IF N1<0
    THEN FOR I:=1 TO -N1 DO Z:=Z/X1;
    CF:=Z;
END;
(• FILE4.....MAIN.PAS •)
(• ..... •)
BEGIN (• MAIN PROGRAM •)
    READLN(X,N);
    Y:=SQR(X)+SH(X)+CF(X,N);
    WRITELN('Y=',Y)
END.

```

INPUT:	OUTPUT
2 5	Y= 3.962686E+01
4 5	Y= 1.067290E+03
4 -5	Y= 4.329090E+01
3 -70	Y= 1.901787E+01

FIG PF20B PROGRAM

这个程序的四个文件，运行步骤基本上与一个文件相同，比外部过程方法要少。其步骤如下：

```

R PASCAL
• MAIN=GLOBAL,SH,CF,MAIN
R MACRO
• MAIN=MAIN
• ^C
R LINK

```

```

• MAIN=MAIN,PASCAL
• AC
• RUN MAIN

```

无论使用外部过程的方法还是使用内部过程的方法，无论内部过程方法中用多个文件还是一个文件，链接后得到的运行程序（SAV 类型文件）的长短总是一样的，即任何方法都不会多占用或节省内存。当然，由于运行步骤有多有少，从时间上是有差异的。

§10 应用举例

过程和函数的应用是非常广泛的，在学习了后面的几章后会体现得更为突出。本节只从巩固概念的角度出发，编制几个简单的应用题，供读者参考。

例5-21 编制程序解二元一次方程组

$$\begin{cases} Ax + By = C \\ Dx + Ey = F \end{cases}$$

要求：解方程组部分用过程语句。

分析：

设 $G = A \cdot E - B \cdot D$ 。

根据代数知识，在 $G = 0$ 时，方程组有无穷多个解。否则，有一组确定的解：

$$x = \frac{1}{G}(C \cdot E - B \cdot F) \quad y = \frac{1}{G}(A \cdot F - C \cdot D)$$

程序如 FIG PF21 PROGRAM 所示。

```

PROGRAM PF21 (INPUT,OUTPUT);
VAR
  A,B,C,D,E,F:INTEGER
  G, X, Y :REAL;
  PROCEDURE SOLVE;
  BEGIN
    G:=A * E - B * D;
    IF      G<>0
    THEN
      BEGIN
        X:=(C * E - B * F)/G;
        Y:=(A * F - C * D)/G;
      END
    END;
  BEGIN      (* MAIN PROGRAM *)
    READLN (A,B,C);
    READLN (D,E,F);
    SOLVE;
    IF      G=0
    THEN WRITELN('THERE ARE MANY SOLUTIONS ')
    ELSE WRITELN('X=',X:6:2, ' , Y=',Y:6:2)

```

```

END.
INPUT :
1      2      3
4      5      6
OUTPUT:
X=-1.00, Y=2.00
INPUT,
1      2      3
4      8      5
OUTPUT:
THERE ARE MANY SOLUTIONS 1
FIG PF21 PROGRAM

```

这是一个比较简单的程序。它只有全程变量，没有局部变量，更没有任何参数。这个程序的执行部分完成三件工作：

1. 输入二元一次方程组的系数值；
2. 解方程组，通过过程语句实现；
3. 输出解方程组的结果。

这个程序稍加修改后，可以解三元一次方程组及其它线性方程组。

例5-22 编制程序解决如下问题：

首先输入某班新生的班号及总人数；

其次输入每个同学各门功课的成绩，并计算该生的总分及平均分；

最后计算该班的总平均分，并输出新生成绩表。

假定该班学生的学号从800001开始，人数不超50人。计算成绩时用过程语句。

程序如 FIG PF22 PROGRAM 所示。

```

PROGRAM PF22 (INPUT,OUTPUT);
LABEL
10;
VAR
M,N,S,A,B,C,D,E,F:INTEGER;
A1,B1,C1,D1,E1,F1:INTEGER;
CLASS :ALFA;
PROCEDURE COUNT (X1,X2,X3,X4,X5,X6:INTEGER);
VAR
SUM:INTEGER; AVER:REAL;
BEGIN
IF M>N
THEN SUM:=S
ELSE
BEGIN
SUM:=X1+X2+X3+X4+X5+X6;
S:=SUM
END;

```



```

    AVER:=SUM/6,
    IF    M>N
    THEN WRITE('TOTAL:')
    ELSE
    IF    M<10
    THEN WRITE('80000',M:1)
    ELSE WRITE('8000',M:2)
    WRITE(X1:5,X2:5,X3:5,X4:5,X5:5,X6:5),
    Writeln(SUM:6,AVER:6:1)
END,
BEGIN    ( • MAIN PROGRAM • )
    READLN(CLASS,N),
    IF    N>50
    THEN
    BEGIN
        Writeln('MORE THAN 50 1 '),
        GOTO 10
    END,
    Writeln('CLASS:':20,CLASS),
    Writeln('NUMBER:':20,N:6),
    FOR M:=1 TO 2 DO Writeln,
    WRITE(' NO.          POLI CHIN MATH PHYS CHEM ENGL'),
    Writeln('SUM':5,'AVERAGE':8),
    FOR M:=1 TO N DO
    BEGIN
        READLN(A,B,C,D,E,F),
        COUNT(A,B,C,D,E,F),
        A1:=A1+A; B1:=B1+B; C1:=C1+C;
        D1:=D1+D; E1:=E1+E; F1:=F1+F;
    END,
    Writeln,
    A:=A1 DIV N; B:=B1 DIV N; C:=C1 DIV N,
    D:=D1 DIV N; E:=E1 DIV N; F:=F1 DIV N,
    COUNT(A,B,C,D,E,F);
10:
    END.
INPUT:
J81          90
OUTPUT:
MORE THAN 50 1
INPUT:
COMPUTE801    10
75    85    90    94    85    79
76    86    91    93    84    78

```

```

.....
95 85 75 76 86 96
OUTPUT:
CLASS:COMPUTE801
NUMBER 10
NO. POLI CHIN MATH PHYS CHEM ENGL SUM AVERAGE
800001 75 85 90 94 85 79 508 84.7
800002 76 86 91 93 84 78 508 84.7
800003 77 87 92 92 83 77 508 84.7
800004 78 88 93 91 82 76 508 84.7
800005 79 89 94 90 81 75 508 84.7
800006 95 89 79 75 85 95 518 86.3
800007 94 89 78 76 84 94 514 85.7
800008 93 87 77 77 83 93 510 85.0
800009 92 86 76 78 82 92 506 84.3
800010 95 85 75 76 86 96 519 85.5
TOTAL: 85 87 84 84 83 85 510 85.0
NOTE:
A-A1-POLI:POLITICS; B-B1-CHIN:CHINESE;
C-C1-MATH:MATHEMATICS; D-D1-PHYS:PHYSICS;
E-E1-CHEM:CHEMISTRY; F-F1-ENGL:ENGLISH;
FIG PF22 PROGRAM

```

在这个程序中，有几点说明如下。

1. 变量说明问题。

ALFA 类型是一种标准数据类型，在第七章中详细介绍，读者可以暂不考虑这个问题；变量 A 到 F 主要表示某个学生各门功课的成绩。最后表示该班各门功课的平均成绩，但是，只取整数。

变量 A1 到 F1 表示该班各门功课成绩的累加和。S 表示该班总分数，都用整数表示。由于人数少于 50 人，因此，总和 S 不会大于 32767。

2. 主程序部分主要步骤：

- (1) 建立学生成绩表表头；
- (2) 逐个输入各位同学的成绩；
- (3) 计算总分及平均分；
- (4) 输出各位同学的成绩（包括总分及平均分）；
- (5) 累加该班各门功课的成绩；
- (6) 输出该班的总平均成绩。

3. 过程 COUNT 的主要步骤：

- (1) 计算和数及平均数；
- (2) 输出学号或总计 (TOTAL)。由于学号由六位数字组成，超过 32767，因此，输出格式必须慎重考虑。例题中通过加字符串办法实现。
- (3) 输出该生或该班各门功课的成绩、总分及平均分。

例5-23 编制程序计算积分 $\int_a^b f(x)dx$ 。

其中 $f(x)=x^2-2x+2$, x 为实数。如图5.9-1所示。 a , b 由用户通过终端键盘输入。积分值通过终端显示器输出。

分析:

$$f(x)=x^2-2x+2=(x-1)^2+1$$

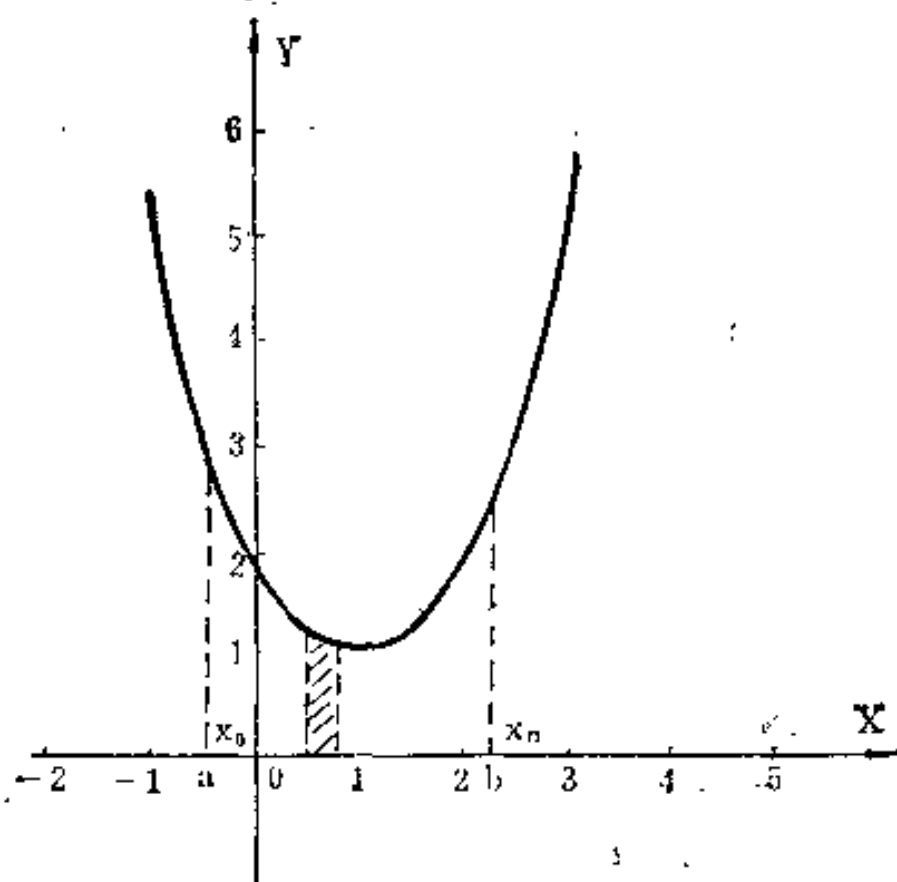


图 5.9-1 $y=(x-1)^2+1$ 函数曲线图

根据数学知识, $\int_a^b f(x)dx$ 等于 $x=a$, $x=b$, $y=0$ 和 $y=f(x)$ 所围成的曲线图形的面积。

为了求得总面积, 先将区间 (a, b) 分成 n 等分, 对应地将图形分成 n 等分。每一个小部分的图形近似于一个小梯形。如阴影部分所示。每个小梯形的高为 $H = \frac{1}{n}(b-a)$ 。

第一个小梯形的面积为:

$$S_1 = \frac{1}{2} H \cdot (f(x_0) + f(x_1))$$

其中 $x_0 = A$, $x_1 = A + H$ 。

第 i 个小梯形的面积为:

$$S_i = \frac{1}{2} H \cdot (f(x_{i-1}) + f(x_i))$$

其中 $x_{i-1} = A + (i-1) \cdot H$, $x_i = A + i \cdot H$

第 n 个小梯形的面积为:

$$S_n = \frac{1}{2} H \cdot (f(x_{n-1}) + f(x_n))$$

其中 $x_n = A + n \cdot H = B$, $x_{n-1} = B - H$

对 n 个小梯形分别求面积, 然后进行累加, 就能得到曲线图形的近似面积。这就是函数积分的基本方法。

在程序设计时， n 由用户确定。

程序如 FIG PF23 PROGRAM 所示。

```
PROGRAM PF23(INPUT,OUTPUT);
VAR
  A,B,V:REAL;
  N :INTEGER;
  FUNCTION F0(X1:REAL):REAL;
  BEGIN
    F0:=SQR(X1)-2*X1+2
  END;
  PROCEDURE JF (FUNCTION F:REAL,VAR P:REAL);
  VAR
    H,X,S,Y1,Y2:REAL;
    I :INTEGER;
  BEGIN
    P:=0;
    H:=(B-A)/N;
    X:=A;    Y1:=F(X);
    FOR I:=1 TO N DO
      BEGIN
        X:=X+H;    Y2:=F(X);
        S:=(Y1+Y2)*H/2;
        Y1:=Y2;
        P:=P+S
      END
    END;
  BEGIN (* MAIN PROGRAM *)
    READLN(A,B,N);
    JF(F0,V);
    WRITELN('V=',V:6:3);
  END.
```

INPUT :			OUTPUT:
-3	3	10	V= 30.360
-3	3	30	V= 30.040
-3	3	100	V= 30.004
-3	3	1000	V= 30.000
10	80	1000	V= 164172.200
10	80	10000	V= 164182.400
3.5	9.9	1000	V= 236.186

FIG PF23 PROGRAM

在这个程序中，过程说明 JF 占有极其重要的位置。它主要包括以下步骤：

1. 计算小梯形的高 H 。 $H:=(B-A)/N$;
2. 计算第一个小梯形的上底 Y_1 。

$$X := A; \quad Y_1 := F(X);$$

3. 计算小梯形的下底 Y_2 。

$$X := X + H; \quad Y_2 := F(X);$$

4. 计算小梯形的面积 S 。

$$S := (Y_1 + Y_2) \cdot H / 2;$$

5. 变第 i 个小梯形的下底 Y_2 为第 $(i+1)$ 个小梯形的上底 Y_1 。

$$Y_1 := Y_2;$$

6. 累加小梯形的面积 P 。

$$P := P + S;$$

7. 重复步骤3—6, 直到累加完毕。

程序运行的结果表明:

1. 区间不同, 积分值不同;

2. N 越大, 积分值越精确。当然, 精确度的提高是靠牺牲运行时间换来的。

按照数学知识, 对于 $f(x) = x^3 - 2x + 2$ 来说,

$$\int_{-1}^3 f(x) dx = \left(\frac{1}{4} x^4 - x^2 + 2x \right) \Big|_{-1}^3 = 30$$

当 $N=10$ 时, $V=30.360$ 。只有 $N=1000$ 时, $V=30.000$ 。

例5-24 编制一个程序, 计算下列函数的积分值。

$$\int_{-1}^{1.5} \sqrt{1-x^2} dx,$$

$$\int_2^{5.3} \frac{1}{\text{LN}(x)} dx,$$

$$\int_1^{2.4} \frac{\text{SIN}(x)}{x} dx,$$

$$\int_{-2}^{3.7} \text{SH}(x) dx.$$

分析:

为了编制程序方便, 分别设下列函数:

$$F_1(x) = \sqrt{1-x^2}$$

$$F_2(x) = \frac{1}{\text{LN}(x)}$$

$$F_3(x) = \frac{\text{SIN}(x)}{x}$$

$$F_4(x) = \text{SH}(x) = \frac{e^x - e^{-x}}{2}$$

由于积分的结果是一个具体的数值, 所以不用过程 JF, 而改用函数 JF, 同时, 在函数 JF 中, 增加数值形式参数 A 和 B。因为积分的区间都比较小, 统一取 $N=1000$ 就可以了。

程序如 FIG PF24 PROGRAM 所示。

```

PROGRAM PF24(INPUT, OUTPUT);
CONST
  N=1000;
FUNCTION F1(X1:REAL):REAL;
BEGIN
  F1:=SQR(1-X1)*SQR(X1)
END;
FUNCTION F2(X1:REAL):REAL;
BEGIN
  F2:=1/LN(X1)
END;

```

```

FUNCTION F3(X1:REAL):REAL;
BEGIN
    F3:=SIN(X1)/X1
END;
FUNCTION F4(X1:REAL):REAL;
BEGIN
    F4:=(EXP(X1)-EXP(-X1))/2
END;
FUNCTION JF(A, B:REAL; FUNCTION F:REAL):REAL;
VAR
    H, S, P, X, Y1, Y2:REAL; I:INTEGER;
BEGIN
    H:=(B-A)/N;
    X:=A;
    Y1:=F(X);
    FOR I:=1 TO N DO
        BEGIN
            X:=X+H;
            Y2:=F(X);
            S:=(Y1+Y2)*H/2;
            Y1:=Y2;
            P:=P+S
        END;
    JF:=P
END;
BEGIN (• MAIN PROGRAM •)
    WRITELN(' JF1=', JF(-1, 0.5, F1));
    WRITELN(' JF2=', JF( 2, 5.2, F2));
    WRITELN(' JF3=', JF( 1, 2.5, F3));
    WRITELN(' JF4=', JF(-2, 3.7, F4))
END.

```

OUTPUT:

JF1= 2.112719E+00

JF2= 2.712179E+00

JF3= 8.324376E-01

JF4= 1.647413E+01

FIG PF24 PROGRAM

在这个程序中，程序说明部分除常量定义外，只有五个函数说明。所有函数都有数值参数，但是函数 JF 有一个函数参数。程序中没有变量说明，这是少有的。

程序执行部分非常简单，只有四个写语句。由于函数 JF 中没有变量参数，数值参数 A, B 的实际参数允许是表达式，比如是常量。

例5-25 用二等分法计算函数 $\sin(x)$ 的零点，要求精度为 10^{-6} ，区间由用户确定。

分析：

二等分法，又叫折半法 (BISECTION)，其基本思想是在区间 $[A, B]$ 中，找出函数 $f(x)=0$ 的 x 值，精度由程序员自己选择， x 值就是函数的零点。主要步骤如下：

1. 设置布尔值 S ， $S := F(A) < 0$;
2. 取 A 和 B 的中点 X ， $X := (A+B)/2.0$;
3. 计算函数值 $F(X)$ ， $Z := F(X)$;
4. 如果 $Z < 0$ 等于 S 则 X 作为新 A ，否则作为新 B 。
 $\text{IF } (Z < 0) = S$
 $\text{THEN } A := X$
 $\text{ELSE } B := X$;

这一步的实质是产生新区间，新区间为老区间的一半（折半法的来历），根据 $F(x)$ 的大小确定中点 X 为新区间的起点或终点。

5. 如果新区间 $[A, B]$ 已经很小，达到了精度要求，那么中点 X 就是零点。否则，重复上述步骤 2，3，4 和 5。

程序如 FIG PF25A PROGRAM 所示。

```

PROGRAM PF25A(INPUT, OUTPUT),
CONST
  EPS=1E-6;
VAR
  M, N, Y:REAL;
  FUNCTION SINE(X1:REAL):REAL;
  BEGIN
    SINE:=SIN(X1)
  END;
  FUNCTION ZERO(FUNCTION F:REAL;A, B:REAL):REAL;
  VAR
    X, Z:REAL;    S:BOOLEAN;
  BEGIN
    S:=F(A)<0;
    REPEAT
      X:=(A+B)/2.0;
      Z:=F(X);
      IF (Z<0)=S
      THEN A:=X
      ELSE B:=X
    UNTIL ABS(A-B)<EPS;
    ZERO:=X
  END;
BEGIN  (• MAIN PROGRAM •)
  READLN(M, N);
  Y:=ZERO(SINE, M, N);
  WRITELN('ZERO=', Y)
END.

```

INPUT:	OUTPUT:
-5 5	ZERO= 3.141592E+00
-5 0	ZERO=-3.141592E+00
-1 1	ZERO=-9.536743E-07
0 5	ZERO= 3.141592E+00
1 2	ZERO= 1.999999E+00
9.5 12	ZERO= 1.200000E+01

FIG PF25A PROGRAM

从这个程序的运行结果来看，由于正弦函数是一个周期函数。因此，在不同的区间会有不同的零点。但是，在一个区间，即使是一个很大的区间，如 $[-5, 5]$ ，通过二等分法最多只能找到一个零点。

此外，必须指出，并不是任何区间都可以找到零点。有的区间内根本就不存在零点。比如在区间 $[1, 2]$ 和 $[9.5, 12]$ 中， $\sin(x)$ 都不可能为零。显然，这个程序将 1.999999 和 12 作为零点是完全错误的。

为了避免这种错误，必须增加一段判断程序。在零点两侧的函数值 $F(A)$ 和 $F(B)$ 应当数值接近，符号相反。如果符号相同，则中点 x 不是真正的零点，而是区间错误的结果。

新程序如 FIG PF25B PROGRAM 所示。

```

PROGRAM PF25B(INPUT, OUTPUT);
CONST
  EPS=1E-6;
VAR
  M, N:REAL;
  FUNCTION SINE(X1:REAL):REAL;
  BEGIN
    SINE:=SIN(X1)
  END;
  PROCEDURE ZERO(FUNCTION F:REAL; A, B:REAL);
  VAR
    X, Z:REAL; S:BOOLEAN;
  BEGIN
    S:=F(A)<0;
    REPEAT
      X:=(A+B)/2.0;
      Z:=F(X);
      IF (Z<0)=S
      THEN A:=X
      ELSE B:=X
    UNTIL ABS(A-B)<EPS;
    IF F(A)*F(B)<=0
    THEN WRITELN('ZERO=', X)
    ELSE WRITELN('THERE IS NO ZERO POINT ;')
  END;

```



```

      BEGIN  (• MAIN PROGRAM •)
      READLN(M, N);
      ZERO(SINE, M, N)
      END.
INPUT:      OUTPUT:
5      -5      ZERO= 3.141592E+00
-5      0      ZERO=-3.141592E+00
-1      1      ZERO=-9.536743E-07
3      -3      ZERO=-7.152557E-07
0      5      ZERO= 3.141592E+00
1      2      THERE IS NO ZERO POINT
9.5    12     THERE IS NO ZERO POINT
      FIG PF25B PROGRAM

```

在新程序中，主要增加了判断中点是否为零点的功能。同时，为了输出计算结果的方便，将函数 ZERO 改成过程 ZERO。

思考题：

如果将程序中的条件 $F(A) * F(B) \leq 0$

改成 $F(A) * F(B) < 0$ 行不行？为什么？

例5-26 编制一个程序，求下列各方程的实根。

$$3x^3 - 7x^2 + x - 5 = 0$$

$$\frac{3e^{-2x}}{4+5x} - 7\ln x = 0$$

$$xe^x - 100 = 0$$

分析：

要解这三个方程是不容易的。牛顿迭代法是解方程的一种方法。求函数零点的方法可以转化为解方程的又一种方法。 $f(x)=0$ ，求出的 x 就是方程的根。

确定方程的根的范围是一件难事。有的方程可以通过分析方程的特点去估计。许多方程则要靠实践去寻找。因此，区间可以由用户在程序运行时确定。

程序如 FIG PF26 PROGRAM 所示

```

      PROGRAM PF26(INPUT, OUTPUT);
      CONST
      EPS=1E-6;
      VAR
      M, N:REAL;
      FUNCTION F1(X1:REAL):REAL;
      BEGIN
      F1:=3 * X1 * SQR(X1) - 7 * SQR(X1) + X1 - 5;
      END;
      FUNCTION F2(X1:REAL):REAL;
      BEGIN
      F2:=3 * EXP(-2 * X1) / (4 + 5 * X1) - 7 * LN(X1);
      END;

```

```

FUNCTION F3(X1:REAL):REAL;
BEGIN
    F3:=X1*EXP(X1)-100
END;
PROCEDURE SOLVE(FUNCTION F:REAL, A, B:REAL);
VAR
    X, Z:REAL;    S:BOOLEAN;
BEGIN
    S:=F(A)<0;
    REPEAT
        X:=(A+B)/2.0;
        Z:=F(X);
        IF (Z<0)=S
            THEN A:=X
            ELSE B:=X
        UNTIL ABS(A-B)<EPS;
    IF F(A)*F(B)<=0
        THEN WRITELN('SOLUTION=', X)
        ELSE WRITELN('THERE IS NO SOLUTION ')
    END;
BEGIN (* MAIN PROGRAM *)
    READLN(M, N);
    WRITE('EQUATION 1: '); SOLVE(F1, M, N);
    WRITE('EQUATION 2: '); SOLVE(F2, M, N);
    WRITE('EQUATION 3: '); SOLVE(F3, M, N)
END
INPUT:          OUTPUT:
1      5        EQUATION 1: SOLUTION=2.471341E+00
              EQUATION 2: SOLUTION=1.006360E+00
              EQUATION 3: SOLUTION=3.385631E+00
0.5    1000     EQUATION 1: SOLUTION=2.471342E+00
              EQUATION 2: SOLUTION=1.006361E+00
              EQUATION 3: EXP OVERFLOW
-1     2        EQUATION 1: THERE IS NO SOLUTION
              EQUATION 2: LOG OF NEGATIVE

```

FIG PF28 PROGRAM

程序运行的结果表明，这三个方程都有一个实根。其中方程 1 还有两个复根。在求出实根后可以求出复根。

如果区间选择不合理，则在那个区间中没有根。比如，在区间 $[-1, 2]$ 中，方程 1 找不到解。

如果区间选择不合理，则会产生运行错误。比如，在区间 $[-1, 2]$ 中，方程 2 出现计算负数的对数问题。(LOG OF NEGATIVE)。又如，在区间 $(0.5, 1000)$ 中，方程 3 出现指数溢出问题。(EXP OVERFLOW)。

思考题:

方程 $x^4 + x^2 + 1 = 0$ 有实根吗?

例5-27 函数递归调用的一个实例。

有一对小兔子, 出生一个月后变成一对大兔子, 两个月后生出第一对小兔子, 自己变成一对老兔子。此时共有两对兔子 (一对老兔子, 一对小兔子)。

三个月后, 老兔子又生出一对小兔子, 上个月生的小兔子变成大兔子。此时共有三对兔子。(老、大、小各一对)。

四个月后, 大兔子变成老兔子, 小兔子变成大兔子, 两对老兔子生出两对小兔子。此时共有五对兔子。(老、小各两对, 大兔子一对)。

.....

请编制程序, 计算某个月后有多少对兔子。

分析:

这种问题在生物学上是不可能发生的。既不可能按照此规律生长出那么多兔子, 更不可能生出品种优良的兔子。

但是, 这在数学上是有意义的。这是求斐波拉契 (FIBONACCI) 数列的方法。其计算公式是:

$$FIB(N) = FIB(N-1) + FIB(N-2)$$

对于本题, 本月兔子数等于上月的兔子数再加前一个月的兔子数。

程序如 FIG PF27 PROGRAM 所示。

```
PROGRAM PF27(OUTPUT);
VAR
  I, N:INTEGER;
FUNCTION FIB(I:INTEGER):INTEGER;
BEGIN
  IF      (I=0) OR (I=1)
  THEN  FIB:=1
  ELSE  FIB:=FIB(I-1)+FIB(I-2)
END;
BEGIN  (* MAIN PROGRAM *)
  READLN(I);
  IF    I<0
  THEN WRITELN('ERROR IN INPUT DATA ')
  ELSE
  BEGIN
    N:=FIB(I);
    WRITELN('  N= ', N:3)
  END
END
INPUT:          OUTPUT:
-5              ERROR IN INPUT DATA
0               N=  1
1               N=  1
```

5 N= 8
10 N= 89
11 N=144
12 N=233

FIG PF27 PROGRAM

在这个程序中，函数 FIB 是直接递归，控制递归调用的条件是月份！，在超过一个月的条件下进行递归调用。

下面是一年内各个月后兔子情况统计表，可供分析问题参考。

I	FIB (I-1)		FIB (I-2)	FIB (I)
月 份	老 兔	大 兔	小 兔	总 数
0	0	0	1	1
1	0	1	0	1
2	1	0	1	2
3	1	1	1	3
4	2	1	2	5
5	3	2	3	8
6	5	3	5	13
7	8	5	8	21
8	13	8	13	34
9	21	13	21	55
10	34	21	34	89
11	55	34	55	144
12	89	55	89	233

例5-28 编制程序完成下列功能：

计算 $Y = \text{TAN}^{-1} \text{COS} \text{TAN}^{-1} \text{COS} \dots \text{TAN}^{-1} \text{COS} \frac{\pi}{3}$ 。

要求精度为 0.5×10^{-6} 。

分析：

本题要求先计算出 $\frac{\pi}{3}$ （即 60° ）的余弦值。再计算该余弦值的反正切值。进一步计算该反正切值的余弦值。……直到计算出的反正切值的精度达到 0.5×10^{-6} 为此。

为此，建立两个过程 P 和过程 Q。

过程 P 计算反正切值。 $B := \text{ARCTAN}(A)$ 。

过程 Q 计算余弦值。 $A := \text{COS}(B)$ 。

两个过程相互间接递归调用。控制递归调用的条件是计算精度。最后输出递归的次数 N 和函数值 Y。

程序如 FIG PF28 PROGRAM 所示。

```

PROGRAM PF28(OUTPUT),
VAR
  A, B:REAL,
  N :INTEGER,
  PROCEDURE P, FORWARD,
  PROCEDURE Q,
  BEGIN
    A:=COS(B),
    P
  END,
  PROCEDURE P,
  BEGIN
    IF ABS(ARCTAN(A)-B)>0.5E-6
    THEN
      BEGIN
        B:=ARCTAN(A),
        N:=N+1,
        Q
      END
    END,
  BEGIN  (• MAIN PROGRAM •)
    B:=3.141592/3,
    Q,
    WRITELN('N=', N:3),
    WRITELN('Y=', B)
  END.
OUTPUT:
N= 15
Y= 6.662392E-01
FIG PF28 PROGRAM

```

在这个程序中，过程P和Q在同一嵌套深度，为了递归调用，采用“向前引用”的方法。在调用15次后，Y为6.662392E-01。

• §11 堆栈技术的应用

前面各节由浅入深地介绍了过程及函数的说明和应用。从中可以初步体会到过程及函数应用的优越性。在学习了后面几章以后，就会进一步体会到过程及函数的重要性及实践性。

为什么过程和函数有这些特点呢？这是因为PASCAL语言采用模块式程序设计方法，这一点在过程和函数中用得最突出。而在过程和函数的设计中，堆栈技术起了极其重要的作用。

在程序设计中，使用过程和函数，可以使程序结构清晰，逻辑关系准确，有利于程序的设计、阅读、调试及修改。

对于多次调用的过程和函数，在程序中只要说明一次。这样数，节省了程序员的设计时间，节省了计算机的存贮空间，易读易懂。

前面的例题主要是为了说明概念，程序往往比较小，实际工作中需要设计的程序常常要大得多，甚至成千上万个语句。这时可以把整个程序划分为几大模块，使每块的功能相对完整独立，同时明确这些模块与主程序之间关系。在程序设计时，可以先设计与调试每一个模块。此时，还可以把它分成几个更小的部分，先分别设计与调试每一个小部分，然后总调这一模块。这就是模块式程序设计方法，也叫结构式程序设计方法。

对于某一个模块，还可以分为不同的层次，外层对内层的功能及调用规则有要求，内层具体解决这些问题，这就是层次模块式程序设计方式，或称为从顶到底的结构式程序设计方法。过程则是实现这种设计方法的有效手段。

堆栈技术在过程和函数的调试与运行中起了极重要的作用。所谓堆栈，是指按“后进先出”的原则进行管理的一个数据存贮区。在 PDP-11 系列机中，是在内存中开辟一个存贮区。用 R6 作为堆栈指针。此时，也可将 R6 记作 SP。

为什么堆栈在过程和函数中如此重要呢？这是因为过程和函数中的局部变量以及递归调用必须用堆栈技术。

从形式上看，过程与程序很相似。但是，程序是在操作系统直接控制下运行，而过程只是程序的一部分，它要经过程序或其它过程调用才能真正投入运行。此外，过程与一般的一段程序（如一个大的复合语句，循环语句，）也不相同。因为当程序运行到这些语句时，这些语句会“自动”开始执行。而对过程来说，则不会如此，它需要转去执行过程说明的执行部分，其中包括参数传递的问题。

局部变量只有在该过程投入运行时才会使用。因此，在这一个过程未投入运行前，不必为它的局部变量分配内存单元。投入运行时，必须为它的局部变量分配内存单元。运行结束后，可以释放局部变量的内存单元，另作它用。这样就可以节省存贮单元。

递归调用时，比如函数递归调用时，每递归调用一次，都要重新分配形式变量单元，局部变量单元和函数结果单元。如果各自只有一个静态的存贮单元，是不能满足递归调用的需要的。在程序设计时，由于递归的次数一般都无法确定。因此，各自事先分配几个静态的存贮单元也是不可能的。

由于递归调用的这些特点，就要求每递归一次，重新分配一次存贮单元。递归结束时，逐层释放所有相应的单元。实现这种动态分配与管理存贮单元的最有效的方式，就是应用堆栈技术。

例5-29 堆栈技术在过程嵌套中的应用。

程序如 FIG PF29 PROGRAM 所示

```
PROGRAM PF29(OUTPUT);  
VAR  
  X, Y, Z:INTEGER;  
  PROCEDURE P1;  
  VAR  
    X, Y:INTEGER;  
    PROCEDURE P2;  
    VAR
```

```

        X:INTEGER;
        BEGIN
            X:=30;
            WRITELN('CALL P2 ; ')
        END;
    BEGIN
        X:=20;
        P2;
        Y:=200;

        WRITELN('CALL P1 ; ')
    END;
BEGIN (• MAIN PROGRAM •)
    X:=10;
    P1;
    Y:=100;
    Z:=1000;
    WRITELN('PROGRAM END ; ')
END.
OUTPUT:
CALL P2 ;
CALL P1 ;
PROGRAM END ;
FIG PF29 PROGRAM

```

在这个程序中，主程序的全程变量为X、Y和Z。过程P1的局部变量为X和Y。过程P2嵌套在过程P1中，它的局部变量为X。根据前面的知识，这三个X的意义是不同的，它们各自的有效范围也不相同：在主程序中，起作用的是全程变量X，过程中的局部变量并不存在（指没有分配存储单元）。在执行过程P1时，起作用的是P1的局部变量X，全程变量X不起作用，但是存在（指占有存储单元），P2的局部变量X不存在。在执行过程P2时，起作用的是P2的局部变量X，全程变量X及P1的局部变量X不起作用，但是都存在。Y的情况与X类似。由于全程变量X是静态地分配存储单元，所以它自始至终都存在。局部变量X是动态地分配存储单元，所以只是使用时它才存在。

下面是本程序运行时，堆栈中存储单元分配的五种情况。

（1）主程序调用P1前。

由于全程变量X、Y和Z不是在堆栈中分配存储单元。因此，堆栈中是空的。

（2）主程序调用P1时。

由于P1有局部变量X和Y，在堆栈中要为它们分配存储单元。因为P1中变量X在Y前面，因此，X在栈底，Y在X上面。

（3）P1调用P2时。

此时，P1仍然起作用，不能从堆栈中撤消它的局部变量。P2有局部变量X，必须为它在堆栈中分配存储单元。

（4）P2调用完毕后。

此时，P2不起作用，则从堆栈中撤消P2的X存贮单元，仍然保留P1的存贮单元。

(5) P1调用完毕后。

此时，P1也不起作用了，可以从堆栈中撤耗P1的X，Y存贮单元。堆栈又变成空栈。堆栈分配的示意图如图5.11-1所示。

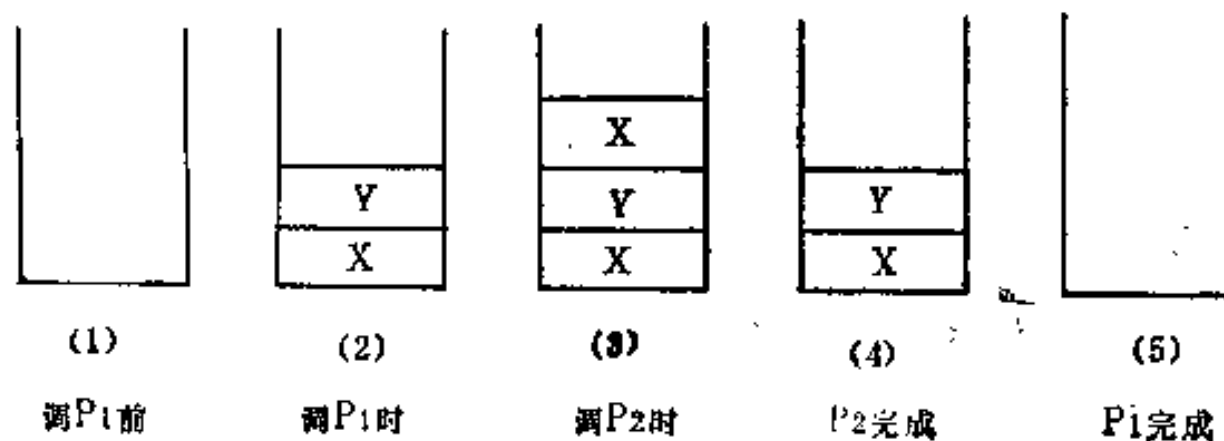


图 5.11-1 堆栈分配示意图

例5-30 堆栈技术在函数递归中的应用

程序如 FIG PF30 PROGRAM 所示。

```

PROGRAM PF30(OUTPUT),
VAR
  I, J: INTEGER,
  FUNCTION JC(X: INTEGER): INTEGER,
  BEGIN
    IF X=0
    THEN JC:=1
    ELSE JC:=X*JC(X-1)
  END,
BEGIN (* MAIN PROGRAM *)
  I:=5;
  J:=JC(I);
  WRITELN('5! =', J:5)
END.
OUTPUT:
5! = 120

```

FIG PF30 PROGRAM

这个程序的功能是计算5的阶乘。计算阶乘的公式是：

$$X! = X \cdot (X-1)!$$

函数JC中又要调用函数JC，这就是递归调用。每递归调用一次，就要给形式参数X，函数JC分配一次存贮单元。递归结束时，逐层释放所有相应单元。

在这个程序中，递归的过程如下：

$$5! = 5 \cdot 4!$$

$$= 5 \cdot 4 \cdot 3!$$

$$= 5 \cdot 4 \cdot 3 \cdot 2!$$

$$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

$$= 120$$

本章习题

1. 输入实数 x ，计算并输出 y ， z 。其中

$$y = \frac{3\operatorname{Sh}4x + 5\cos(3x-5)}{4\operatorname{CH}(3x+5)}, \quad z = \frac{\arcsin(2x+15)}{\operatorname{SH}3x \cdot \operatorname{CH}(-3x)}$$

2. 输入实数 u ， v ，计算并输出 z 值

$$z = \frac{F(u, v)}{F(u+v, u/v) + F(3u+2v, 3u-2v)}$$

$$\text{而函数 } f(x, y) = \frac{2\sin^2(2x+y) - 3\sin(x+y)\cos(x-y)}{e^x \cdot \ln|x| \cdot \operatorname{CH}\frac{x}{2}}$$

3. 计算贝塞尔 (BESSEL) 函数的积分

$$J(x) = \frac{1}{\pi} \int_0^\pi \cos(t - xsint) dt$$

其中 $x = 0, 0.5, 1.0, \dots, 5.5$

4. 指出下列程序中的全程变量、局部变量、变量参数、数值参数，写出程序运行后的输出结果。

```
PROGRAM PFOC(OUTPUT);
VAR A, B, C: INTEGER;
PROCEDURE P(VAR X: INTEGER, Y: INTEGER);
VAR M, N: INTEGER;
BEGIN
    M := X * Y; X := X + 5; Y := Y + 5; N := X * Y;
    Writeln('X=', X:3, ' Y=', Y:3, ' M=', M:4, ' N=', N:4)
END;
BEGIN
    A := 3; B := 3;
    P(A, B); P(A, B); P(A, B)
END;
```

5. 对下列程序中的错误进行编号，指出每个错误的性质。

```
PROGRAM TYPE (INPUT, OUTPUT);
LABEL 10, 20;
VAR I, J: INTEGER; R: REAL;
PROCEDURE PRINT (VAR X: INTEGER, Y: REAL)
VAR Z: REAL;
BEGIN
    IF X < 0 THEN GOTO 10;
    Z := X * Y;
    Writeln(X, Y, Z);
    10:
```

```

END;
BEGIN
    READLN(R);
    PRINT(15, R);
    PRINT(15, 15);
    PRINT(R, R);
    I:=ROUND(R);
    PRINT(I, 0);
    / WRITELN(1:7:2, R:7:2)
END

```

6. 这一章第十节中例5-27所示的题目——函数递归调用实例。只是把题目的要求改为：列表输出自第十二月之后起一年内每个月的兔子总数。

7. 把第十一章第三个程序中的五个内部过程——五种排序方法，改为用外部过程实现，并且在运行时用复盖链接法处理。

第六章 用户自定义数据类型

本书第二章概括地介绍了 PASCAL 语言的数据类型，并详细地讨论了四种标准的数据类型：整数类型，实数类型，字符类型和布尔类型。它们各自的特性已由 PASCAL 语言完全确定，用户可以直接运用它们去分析和解决问题。

从本章开始，本书将用四章的篇幅讨论其余的数据类型。这些类型的数据，完全由用户自己依据 PASCAL 语言的语法规则去确定，即进行类型定义，通过类型定义，进一步确定它们各自的特性。

PASCAL 语言的程序结构规定，类型定义应该在常量定义之后，变量说明之前。这样定义的数据类型适用于整个程序。反之，在程序的过程说明或函数说明中定义的某些数据类型，仅仅适用于那个过程或函数的内部。

这些新的数据类型的定义，大大增强了 PASCAL 语言的功能。它充分显示了 PASCAL 语言的优点，是 PASCAL 语言具有的特色之处。

类型定义的优点主要体现在两方面。

一方面，用这些新的类型描述某些问题会更清楚，而且更便于书写和阅读。

另一方面，类型定义向编译程序提供了更多的信息，编译时就能用这些信息进行多种查错。从而得到更高质量的程序。

在这一章仅仅介绍两种用户自定义数据类型——枚举类型和子界类型。它们都属于简单数据类型。它们所包含的数据通常被称为这种类型的元素。

§1. 枚举类型

一、问题的提出 为什么要使用枚举类型的数据呢？在程序设计的过程中，常常要用到某些很难用四种标准类型表示的数据。比如，怎样才能最方便最清楚地表示一个星期中的每一天，一年中的每一个月呢？又比如，怎样才能最方便最清楚地表示一个班中每一个学生，以及他们在一年中所学的每门功课呢？对于前一类问题，很容易想到，可以用每一天在一周中先后的次序来表示，如用整数 1 表示星期一，整数 2 表示星期二，……用整数 0 表示星期日。对一年中每一个月也可以用类似的办法。这种办法虽然可行，但不够清楚，因为整数 1 不仅能表示星期一，也可以表示一月份，还可以表示一个别的什么东西。它与星期一、一月份的对应关系是每个程序员来约定并使用的，容易出错。要让别人了解这一关系，也要特殊加以说明。人们希望不用整数来代表它们，而直接用人类语言中正常使用的星期一、星期二、一月、二月来表示。对第二类问题更复杂一点，因为每个班中的学生可以变化，他们之间也不存在大家一致公认的排序关系，对于他们所学的每门功课也一样。如果还要用某些整数作为序号来表示每个学生、每门功课，序号与他们之间的对应关系更不好确定了。其实，表示每个学生最简单而有效的办法是用每个人的姓名，表示每门功课也可以用自然语言中每门功课的名称。这样做更自然更清楚，问题是有可能呢？

上面的例子是生活中常见到的，容易理解。在编写编译程序、操作系统的过程中，这类

例子就更多了。例如，在编译程序中，我们可以把一个标识符分成类型标识符，常量标识符，过程标识符，变量标识符，记录变量中的域标识符等；可以把类型的形成分成数字型，符号型，指针型，集合型，数组型，记录型，文件型等；可以把标识符分成实际参数标识符，形式参数标识符两种，如此等等，不一一列举。在操作系统程序中，我们可以把系统状态分成管态与用户态两种，可以把一个进程的状态分成就绪，挂起，运行三类，可以把设备分成系统设备与用户设备两类等等。如果我们把这一切也用一组编号来表示与区分的话，也不是一个好的方法。

综上所述，要能最简单清楚地更自然地表示上述各类问题，就要允许用户本身使用尽可能接近自然语言的方式，来描述这类问题，而不是用经过某种转换，把本来不是简单地用整数来描述的问题硬用几个整数去描述去解决。PASCAL 语言中的枚举类型就是为了解决这类问题而设立的。

要解决上述问题，应有些什么样的要求呢？

第一，每一个具体的枚举类型的名字与它所包括的每一个数据，都应该是一个标识符，并且由用户按需要选定（用户自定义的含意）；这样，用户才能用这种类型描述各类问题；

第二，在定义一个具体的类型时，必须给出它的全部数据。也就是说，用户应明确地列举出这一类型所包括的全部数据（这就是枚举的含意）；

第三，在一个类型的全部数据之间，应该存在某种确定的关系，如大小关系，先后关系，这样才可以在这些数据之间进行比较、寻找等操作（如星期一的下一天是星期二，从一月可以逐月寻找到十二月等）。

枚举类型的数据是满足这些要求的。

二、枚举类型的定义与使用 枚举类型是一种简单的非标准数据类型。它通过枚举一系列有序的标识符来定义。这些标识符由用户根据需要自己确定。它们各自代表一个数据值，称为枚举类型的元素。它们之间有先后顺序，可以进行比较。

枚举类型定义的格式如图 6.1-1 所示：

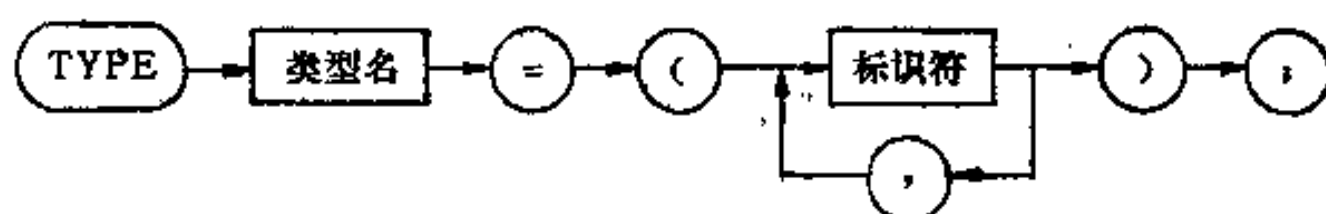


图 6.1-1 枚举类型定义的格式

其中，

类型名和标识符由用户自己确定，但必须符合 PASCAL 语言关于用户标识符的规定。

例如，前面提到的一个星期中有星期日，星期一，……星期六，可以定义类型 WEEKDAYS 如下：

TYPE

WEEKDAYS=(SUN, MON, TUE, WED, THU, FRI, SAT);

在定义了类型 WEEKDAYS 之后，则在四种标准数据类型 INTEGER, REAL, CHAR, BOOLEAN 之外，又有了 WEEKDAYS 可用。如果变量 WORK 和 REST 属于 WEEKDAYS 类型，则可以进行如下变量说明：

VAR

WORK, REST: WEEKDAYS;

变量 WORK (工作日) 和 REST (休息日) 只能表示七天中的某一天, 而不能是几天。例如, 星期日是休息日, 则可以用赋值语句表示如下:

REST := SUN;

现在可以把前面提到的每一问题定义如下:

TYPE

MONTH = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC);

STUDENTS = (CHANG, WANG, LI, ZHAO, LIU, SUN, ...);

COURSE = (MATHEMATICS, PHYSICS, CHEMISTRY, POLITICS, ...);

IDCLASS = (LABEL, CONST, TYPE, VAR, PROCEDURE, FIELD, ...);

TYPEFORM = (NUMERIC, SYMBOLIC, SET, ARRAY, RECORD, FILE, POINTER);

IDKIND = (ACTUAL, FORMAL);

STATUS = (COMMAND, MONITOR, USER);

DEVICE = (SYSTEM, USERS);

PASCAL 语言允许将类型定义和变量说明加以合并, 通过一个变量说明来实现。例如:

VAR

X: (A, B, C, D, E, F);

Y: (X1, Y2, Z3);

这时省去了类型定义, 但是类型的内容且没有省去。它们表示, X 是枚举类型变量, 它可以是六个元素 (A, B, C, D, E, F) 中的任何一个。Y 是枚举类型变量, 它可以是三个元素 (X1, Y2, Z3) 中的任何一个。

必须指出, 这里的 A, B, C, D, E, F 是用户自己定义的标识符, 每个都代表一个数据, 它们与字符 'A', 'B', 'C', 'D', 'E', 'F' 意义完全不同。请读者不要对此产生误解。

枚举类型数据只能进行两种运算: 赋值运算和关系运算。它不能进行算术运算或逻辑运算。这是枚举类型数据的一个特点。

在赋值运算时, 变量名与数据值必须严格地属于同一种枚举类型, 不允许张冠李戴, 混合赋值。

在关系运算时, 是根据它们在类型定义时的顺序进行比较, 而得到布尔型结果。六种常用的关系运算符都适用于枚举类型。

此外, 有三个标准函数的自变量允许属于枚举类型。它们是前导函数 (PRED), 后续函数 (SUCC) 和序数函数 (ORD)。

前两个函数的函数值是枚举类型数据。例如前页定义的 WEEKDAYS 中

PRED (MON) 为 SUN;

SUCC (THU) 为 FRI;

序数函数的函数值为一个整数, 而且第一个枚举类型数据的序数值规定为 0, 以后顺序加 1。例如:

ORD (SUN) 为 0;

ORD (SAT) 为 6;

当然，第一个枚举型元素没有前导值，最后一个枚举型元素没有后续值。

例6-1 定义枚举类型 DAYS，进行简单的赋值运算、关系运算及函数计算。

程序如 FIG USE01 PROGRAM 所示。

```
PROGRAM USE01(OUTPUT);
TYPE
  DAYS=(SUN, MON, TUE, WED, THU, FRI, SAT);
VAR
  I, J:INTEGER;
  A, B:DAYS;
  D:BOOLEAN;
BEGIN
  A:=MON;    B:=PRED(A);
  D:=A<B;    WRITELN('D IS',D);
  I:=ORD(A); WRITELN('I IS',I:6);
  J:=ORD(B); WRITELN('J IS',J:6)
END.
OUTPUT:
D IS FALSE
I IS      1
J IS      0
```

FIG USE01 PROGRAM

例6-2 枚举类型是一种很有使用价值的数据类型。但是在使用过程中也容易出错，出错的原因通常是用户的要求超越了枚举类型的可能性。

FIG USE02 PROGRAM 是一个有九个错误的程序，错误的性质在程序中用(*...*)表示。

```
PROGRAM USE02 (OUTPUT);
CONST
  REST=SUN;
  (* BAD CONSTANT *)
TYPE
  DAYS=(SUN, MON, TUE, WED, THU, FRI, SAT);
  R=(A1, B2, C3, D4, E5, F6, SIN);
  S=('A', 'BC', '5', '9', 10, 2.5);
  (* BAD SCALAR TYPE *)
  T=(ARRAY, RECORD);
  (* BAD SCALAR TYPE *)
  M=(1A, 2B, #, ?, A B);
  (* BAD SCALAR TYPE *)
  (* INVALID CHARACTER *)
  N=(M, N, P, Q, SUN);
  (* BAD SCALAR TYPE *)
VAR
  A, B, C:DAYS;
```

```

D:BOOLEAN;
Y:REAL;
Z:R;
BEGIN
  Z:=SIN;
  Y:=SIN(1.57);
    (*MISSING OPERATOR*)
  READLN(C);
    (*BAD READ STATEMENT*)
  WRITELN(Z);
    (*BAD WRITE STATEMENT*)
END.

```

FIG USE02 PROGRAM

这些错误是常见的，现在分析如下：

1. 由于常量定义优先于类型定义，因此类型定义时确定的数据值（如枚举类型的值 SUN）不能作为常量定义。否则，编译时指出“常量有错”（BAD CONSTANT）。

2. 枚举类型数据必须符合用户标识符的规定。标识符必须字母开头，后面可以是字母或数字。因此，# 或 ? 等字母、数字以外的字符作为“无效的字符”（INVALID CHARACTER），10，2.5，'A'，'9'，1A 等标识符作为“枚举类型有错”（BAD SCALAR TYPE）。

3. 标识符严禁使用 PASCAL 语言的保留关键字。如 ARRAY，RECORD 等。否则，编译时指出“枚举类型有错”。

4. 标识符只是在个别特殊情况下允许为标准标识符。例如，在程序中把 SIN 作为枚举类型 R 的一个数据，把 Z 说明为 R 类型的一个变量。这时，Z:=SIN 是合法的，但是 Y:=SIN (1.57) 则是非法的。编译时指出“遗漏操作符”（MISSING OPERATOR）。原因在于这里的 SIN 已经失去正弦函数的意义。SIN (1.57) 不能表示任何数据，它已变为非法的了。

为避免不必要的麻烦与混乱，用户标识符原则上禁止使用 PASCAL 语言的标准标识符。

5. 标识符不允许双重定义。例如 SUN 是 DAYS 类型的一个数据，M 是另一个枚举类型的名称，都不能再作为新的枚举类型 N 的数据，否则，那样定义的类型 N 是“枚举类型有错”。

6. 枚举类型变量不允许直接通过读语句输入，也不能通过写语句输出。因此，在编译时，指出 READLN (C) 为“读语句有错”（BAD READ STATEMENT），指示 WRITELN (Z) 为“写语句有错”（BAD WRITE STATEMENT）。

枚举类型数据虽然不能直接输入或输出，但是，它的应用仍然是很广泛的。在流程控制方面，它能作为条件语句的条件和重复语句的控制变量。

例6-3 编制一个程序，将枚举类型应用到流程控制中去。

程序如 FIG USE03 PROGRAM 所示：

```

PROGRAM USE03 (OUTPUT);
TYPE

```

```

    DAYS=(SUN, MON, TUE, WED, THU, FRI, SAT);
VAR
    I:INTEGER;
    WORK:DAYS;
BEGIN
    WRITELN ('LOCAL ORDER:');
    WORK:=MON;
    WHILE WORK<=SAT DO
        BEGIN
            WORK:=SUCC (WORK);
            I:=ORD (WORK);
            WRITE (I:3)
        END;
    WRITELN;
    WRITELN;
    WRITELN ('WORK CASE :');
    FOR WORK:=SUN TO SAT DO
        CASE WORK OF
            MON:WRITELN (' 1 .....TEACH COMPUTER1 ');
            TUE:WRITELN (' 2 .....TEACH PROGRAMMING1 ');
            WED:WRITELN (' 3 .....TEACH COMPUTER1 ');
            THU:WRITELN (' 4 .....TEACH PROGRAMMING1 ');
            FRI:WRITELN (' 5 .....STUDY POLITICS1 ');
            SAT:WRITELN (' 6 .....STUDY ENGLISH1 ');
            ELSE WRITELN (' 0 .....OPEN1 ')
        END
    END.
END.

```

OUTPUT:

LOCAL ORDER:

2 3 4 5 6 7

WORK CASE:

0OPEN₁

1TEACH COMPUTER₁

2TEACH PROGRAMMING₁

3TEACH COMPUTER₁

4TEACH PROGRAMMING₁

5STUDY POLITICS₁

6STUDY ENGLISH₁

FIG USE03 PROGRAM

在这个程序中，主要用当语句、循环语句和情况语句，其次是赋值语句和写语句。

在当语句中，不断改变变量 WORK 的值，它是通过计算后续函数的方法实现的。在变量 WORK 不大于 SAT（星期六）时，继续不断循环。而 WORK=SAT 时，SAT 没有后续值，找不到数据 SUCC (SAT)，但是，它的序数值依然存在，这就是序数值中有 7 的原

因。

枚举类型变量能作为循环语句的控制变量，枚举类型数据能作为循环语句的初值和终值。使用情况与整数或字符相同。

情况语句的选择表达式是枚举类型变量，情况标号是枚举类型数据。这个程序中表示的是一个教师从星期日到星期六工作的简单情况：

星期一、三：讲计算机原理课

星期二、四：讲程序设计课

星期五：学习政治

星期六：进修英语

星期日：自由地安排工作

例6-4 一个枚举类型M由A、B、C、D、E和F组成。程序要求输入数据，计算并输出其序数值和后续值，进行关系运算，输出比较结果

分析：

由于枚举类型数据不能直接输入和输出，必须通过转换的过程来间接地输入和输出。

对于这个问题，可以采用下述步骤：

1. 输入字符。
2. 将字符转换成对应的枚举类型数据。
3. 对枚举类型数据进行赋值运算、关系运算和函数计算。
4. 将枚举类型数据转换成对应的字符输出。

程序如 FIG USE04A PROGRAM 所示。

```
PROGRAM USE04A (INPUT, OUTPUT);
TYPE
  M=(A, B, C, D, E, F);
VAR
  I:INTEGER;
  Z:BOOLEAN;
  CH:CHAR;
  X,Y:M;
PROCEDURE INM (CIN:CHAR,VAR XIN:M);
BEGIN
  CASE CIN OF
    'A' : XIN:=A;
    'B' : XIN:=B;
    'C' : XIN:=C;
    'D' : XIN:=D;
    'E' : XIN:=E;
    'F' : XIN:=F;
  END;
END;
PROCEDURE OUTM (YOUT:M,VAR COUT:CHAR);
BEGIN
  COUT:=CHR (ORD (YOUT) +65)
```

```

    END;
BEGIN
    READLN;
    READLN (CH);
    INM (CH, X);
    I:=ORD (X);
    WRITELN ('I =', I:3);
    Y:=SUCC (X);
    OUTM (Y, CH);
    WRITELN ('Y =', CH:3);
    Z:=Y>C;
    WRITELN ('Z =', Z);
END
INPUT:   OUTPUT:
A        I =  0
          Y =  B
          Z = FALSE
D        I =  3
          Y =  E
          Z =  TRUE
F        I =  5
          Y =  G
          Z =  TRUE
?        I =  0
          Y =  B
          Z = FALSE

```

FIG USE04A PROGRAM

在这个程序中，枚举类型数据的一个重要特点是由单个字母组成。因此，它在枚举类型 M 中的序数值与其对应的字符在 ASCII 码中的序数值相差 65，这就为枚举类型数据转换为字符输出提供了极大的方便。但是，从序数值找对应的枚举类型数据是无法通过函数直接进行，通常由情况语句来完成。

仔细观察输入和输出情况就会发现，如果输入的字符没有对应的枚举类型数据，则计算和输出的结果都是错误的。输入问号 ? 时就是这种情况。

为了解决这个问题，适当修改过程 INM，在情况语句中加上 ELSE 语句。

程序如 FIG USE04B PROGRAM 所示。

```

PROGRAM USE04B (INPUT, OUTPUT),
LABEL
    10;
TYPE
    M=(A, B, C, D, E, F);
VAR
    I:INTEGER;
    Z:BOOLEAN;

```

```

CH:CHAR;
X,Y:M;
PROCEDURE INM(CIN:CHAR,VAR XIN:M);
BEGIN
    CASE CIN OF
        'A' : XIN:=A;
        'B' : XIN:=B;
        'C' : XIN:=C;
        'D' : XIN:=D;
        'E' : XIN:=E;
        'F' : XIN:=F;
    ELSE
        BEGIN
            WRITELN('SCALAR ERROR ');
            GOTO 10;
        END;
    END;
END;
PROCEDURE OUTM(YOUT:M,VAR COUT:CHAR)
BEGIN
    COUT:=CHR(ORD(YOUT)+65)
END;
BEGIN
    READLN;
    READLN(CH);
    INM(CH, X);
    I:=ORD(X);
    WRITELN(' I =', I:3);
    Y:=SUCC(X);
    OUTM(Y, CH);
    WRITELN(' Y =', CH:3);
    Z:=Y>C;
    WRITELN(' Z =', Z);
10:
    END
INPUT.  OUTPUT:
  A      I =  0
         Y =  B
         Z = FALSE
  D      I =  3
         Y =  E
         Z =  TRUE
  F      I =  5
         Y =  G

```

```

          Z = TRUE
          SCALAR ERROR;
6          SCALAR ERROR;
FIG USE04B PROGRAM

```

例6-5 枚举类型 DAYS 由星期日到星期六组成。间接输入今天是星期几，计算并输出昨天和明天是星期几。

分析：

DAYS 型数据由多个字母组成，不能简单地照搬上一题的方法。

为了输入的方便，通过一个过程将输入的整数转换成枚举类型数据。这些整数作为枚举类型数据的序数值。如果输入整数大于6或小于0则作为“枚举错误”处理。

为了输出方便，通过一个过程将枚举类型数据转换成字符串输出。

对于昨天和明天的情况，考虑到实际生活中，星期日的前一天是星期六，星期六的下一天是星期日。程序中用两个简单的过程来处理，不是笼统地用前导函数和后续函数来解决。

程序如 FIG USE05 PROGRAM 所示。

```

PROGRAM USE05 (INPUT, OUTPUT),
LABEL
10;
TYPE
  DAYS=(SUN, MON, TUE, WED, THU, FRI, SAT);
VAR
  K:INTEGER;
  D D1 D2:DAYS;
  PROCEDURE INDAY (KIN:INTEGER; VAR DIN:DAYS);
  BEGIN
    CASE KIN OF
      0 : DIN:=SUN;
      1 : DIN:=MON;
      2 : DIN:=TUE;
      3 : DIN:=WED;
      4 : DIN:=THU;
      5 : DIN:=FRI;
      6 : DIN:=SAT;
    ELSE
      BEGIN
        WRITELN ('SCALAR ERROR ');
        GOTO 10
      END
    END
  END;
  PROCEDURE OUTDAY (DOUT:DAYS);
  BEGIN
    CASE DOUT OF
      SUN : WRITELN ('SUN');

```

```

    MON      :   WRITELN ('MON');
    TUE      :   WRITELN ('TUE');
    WED      :   WRITELN ('WED');
    THU      :   WRITELN ('THU');
    FRI      :   WRITELN ('FRI');
    SAT      :   WRITELN ('SAT')
END
END;
PROCEDURE LASTDAY (TODAY: DAYS; VAR LDAY: DAYS);
BEGIN
    IF TODAY=SUN
    THEN LDAY:=SAT
    ELSE LDAY:=PRED (TODAY)
END;
PROCEDURE NEXTDAY (TODAY: DAYS; VAR NDAY: DAYS);
BEGIN
    IF TODAY=SAT
    THEN NDAY:=SUN
    ELSE NDAY:=SUCC (TODAY)
END;
BEGIN (* MAIN PROGRAM *)
    READLN (K);
    INDAY (K, D);
    WRITE ('TODAY :');
    OUTDAY (D);
    LASTDAY (D, D1);
    WRITE ('YESTERDAY:');
    OUTDAY (D1);
    NEXTDAY (D, D2);
    WRITE ('TOMORROW :');
    OUTDAY (D2);
10;
END.
INPUT:      OUTPUT:
1          TODAY      :MON
           YESTERDAY :SUN
           TOMORROW  :TUE
6          TODAY      :SAT
           YESTERDAY :FRI
           TOMORROW  :SUN
0          TODAY      :SUN
           YESTERDAY :SAT
           TOMORROW  :MON
3          TODAY      :WED

```

```

YESTERDAY:TUE
TOMORROW:THU
-1      SCALAR ERROR;
8       SCALAR ERROR;
FIG USE05 PROGRAM

```

§2. 子界类型

在一些程序中,某些变量的变化范围,可能只局限在某一类型的全部数值中的某一个确定的区域内,如人的年龄,不会取负值,也还未发现超过150岁的,具体到某一个单位,很可能找不到100岁的老人。所以在定义一个AGE变量时,可以规定其变化范围为0~100,显然0~100是整数类型数据的一部分。

为什么要规定AGE为0~100,而不把AGE说明为整型变量呢?原因有三:

1. 把人的年龄说明为0~100,更符合人的常规概念,因此看起来更自然更清楚;
2. 在绝大多数的PASCAL编译程序生成的目标程序中,都给出对子界类型变量的上下界检查,这可以检查出程序中出现的某些错误,如计算某一个人年龄时,若结果为负,或大于100,就能发现这是一个程序错误,可以向程序员提供指示信息,程序员就可以检查自己的程序,更容易找出问题,避免一些错误结果。
3. 在使用有PACK与UNPACK过程的编译程序时,可以用PACK过程处理某些子界类型变量,以节省所占内存容量。如为了表示0~100,只要用一个字节就够了,而不是两个或更多的字节,如果一个程序中包括相当多这类数据,那么节省的内存容量就很可观的了,当然要多占用一点运行时间。

从上述三点可以看出,在设计一个程序时,凡是在可以使用子界类型的地方应尽量使用子界类型。这很方便,然而得到的好处是明显的。

子界类型是另一个简单的非标准数据类型。它通过两个常量来定义,这些常量由用户根据PASCAL语言的语法规则来确定。

子界类型定义的格式如图6.2-1,



图 6.2-1 子界类型定义的格式

其中:

类型名由用户自己确定,但必须符合用户标识符的规定。常量1和常量2必须是两个属于同一类型的有序的数据,常用的是整数和字符,已经定义为枚举类型的数据也可以使用。布尔类型只有两个数,不必要形成子界类型。实数是无序的数据,不能来用组成子界类型。

常量1和常量2通常称为下界和上界,下界必须小于上界。

例如,下面的类型定义都是正确的。

TYPE

```

DAYS=(SUN, MON, TUE, WED, THU, FRI, SAT);
YEAR=1981..2000;

```

```
DIGIT='0'..'9';  
LETTER='A'..'Z';  
WORK=MON..SAT;
```

它们分别表示:

DAYS 类型为枚举类型, 包含 7 个元素。

YEAR, DIGIT, LETTER, WORK 均为子界类型。

YEAR 类型以 1981 到 2000, 共 20 个元素。

DIGIT 类型从字符 0 到字符 9, 共包含 10 个元素。

LETTER 类型从字符 A 到字符 Z, 共包含 26 个元素。

WORK 类型从星期一到星期六, 包含 6 个元素。

但是, 下面的子界类型定义都是错误的:

```
TYPE
```

```
    P = 1500..1000;
```

```
    Q = 1.53..10.6;
```

```
    R = MON..SAT;
```

这是因为 P 的下界大于上界; 实数不能作为子界类型的常量; MON 和 SAT 在类型定义时是未知数, 不能作为常量。

与枚举类型相类似, 子界类型的定义和子界变量的说明可以进行合并。例如:

```
VAR
```

```
    X: 1..10;
```

```
    Y: 'A'..'F';
```

然而, 在大多数情况下, 将类型定义和变量说明分开来更好。

思考题:

下述类型定义是否正确? 为什么?

```
TYPE
```

```
    SUBCHAR='0'..'Z';
```

```
    M=(A, B, C, D, E, F);
```

```
    N=A..D;
```

关于子界类型变量的运算, 一般类似于子界的常量数据类型的运算。

适用于子界的常量数据类型的任何运算符同样适用于子界类型变量。

例如, 对于常量为整数的子界变量, 可以进行算术运算、关系运算和赋值运算。对于常量为字符或枚举型数据的子界变量, 只允许进行关系运算和赋值运算。

如果两个子界类型变量的常量类型相同, 那么, 可以用于赋值语句的两边。例如:

```
VAR
```

```
    I: 1..10;    J: 1..100;
```

它们的常量都是整数类型。下面两个赋值语句都是允许的, 它们是:

```
    J:=I;
```

```
    I:=J+5;
```

但必须指出, 在对子界类型变量进行运算的过程中, 应防止计算结果超出给定的范围, 否则就是一个错误。我们已在本章开始部分说明使用子界类型的理由时就解释了这一问题,

不少参考书对这一问题讨论得比较多，我们不准备多做说明。顺便提一句，在我们所使用的 OMSI PASCAL-1 编译版本中，没有对子界类型的上下界进行检查的功能，因此，程序员更要在自己的程序中多加注意。

在同一个表达式中，可以混合使用不同的子界类型，甚至与其它类型混合，但必须符合基本的语法规则。例如：

```
VAR  
    I : 1 .. 10;      J : 1 .. 100;  
    K : INTEGER;      CH : 'A' .. 'F';
```

那么语句：

```
K := K + J DIV I + ORD(CH) + 150;
```

是合法的。

子界类型变量同样可以作为函数的自变量，函数的选择取决于子界的常量的类型——整数类型、字符类型或枚举类型。

必须指出，当子界的常量为枚举类型数据时，对子界类型变量求序数函数值，给出的是相应枚举类型数据本身的序号。例如：

```
VAR  
    WORK : SUN .. SAT;  
    K : INTEGER;  
BEGIN  
    WORK := FRI;  
    K := ORD(WORK);  
    WRITELN('K = ', K: 3);  
    .  
    .  
    .
```

在程序执行后，显示结果，

K = 5

大家知道，枚举类型数据 WORK 的元素 FRI 在枚举类型 SUN .. SAT 中的排列序号就是 5。

但是，如果子界的常量为字符类型数据，那么子界变量的序数值就是 ASCII 码表中的字符码。例如：

```
VAR  
    CH : 'A' .. 'F';      K : INTEGER;  
BEGIN  
    CH := 'B';  
    K := ORD(CH);  
    WRITELN('K = ', K: 3);  
    .  
    .  
    .
```


在程序执行后，显示结果：

K = _66 (字符B的ASCII码是十进制的66)

如果子界的常量是整数或字符，那么可以直接通过读、写语句对子界变量进行输入输出。

例如：

READLN (I, J);

WRITELN(CH);

都是允许的语句。

如果子界的常量是枚举类型数据，那么不允许通过读写语句对子界变量进行输入输出。

例如，上述例题中如果有语句：

READLN(WORK); WRITELN(WORK);

这是不允许的。编译时会显示“读语句有错”和“写语句有错”。

例6-6 输入不同类型的子界类型数据，对它们进行赋值运算和函数运算。最后输出全部运算结果。

程序如FIG USE06 PROGRAM所示。

```
PROGRAM USE06(INPUT,OUTPUT);
VAR
  I :1..10;           J:1..100;
  K :INTEGER;         C:CHAR;
  CH:'A'..'F';
BEGIN
  READLN(I,J);
  READLN(CH);
  K:=J DIV I+ORD(CH)+100;
  WRITELN('K=',K:6);
  J:=K;
  WRITELN('J=',J:6);
  I:=SQR(J);
  WRITELN('I=',I:6);
  C:='Q';
  CH:=PRED(C);
  WRITELN('CH=',' ',CH,' ');
END.
INPUT:
5 16
B
OUTPUT:
K = 169
J = 169
I = 28561
CH='P'
INPUT,
50 160
```

```

      ?
      OUTPUT,
      K = 166
      J = 166
      I = 27556
      CH = 'P'
      FIG USE06 PROGRAM

```

这个程序在运行时，输入的数据超过了子界范围。如 I 为 50，J 为 160，CH 为 'P'，它们照常运行。运算的结果超过子界范围。如 J = 169，CH = 'P'，它们照常输出。原因在于 OMSI PASCAL-1 不对子界类型变量进行上下界检查，这是程序设计中的错误。

回顾一下标准数据类型中的整数类型和字符类型，它们实质上也是一种特殊的子界类型。其说明方法如下：

```

TYPE
  INTEGER = -32768..32767;
              (对于PDP11/03机器来说)

```

```

  CHAR = ' '.. '~';

```

它们表示，整数型数据的下界为 -32768，上界为 32767，包括整数零共有 65536 个整数。字符型数据的下界为空格字符，上界为 '~' 字符，共有 95 个字符，当然，由于它们是 PASCAL 语言本身定义的，所以在使用时更加简单，不必要进一步说明。

OMSI PASCAL-1 语言规定：

标准的 INTEGER 类型子界为 -32768..32767。因此在 PDP-11 系列机上可以进行有符号位的算术运算。也可以通过子界 0..65535 说明一个无符号整数类型，并规定不对这类变量的乘除等运算进行溢出检查。我们可以用这类变量表示内存地址等。顺便提一句，OMSI PASCAL-1 语言没有安排专门的过程对这类变量进行输出处理。

例 6-7 输入整数 I；计算 $J = 2 \times I$ ；输出 J。其中 J 定义为无符号整数。

程序如 FIG USE07A PROGRAM 所示。

```

      PROGRAM USE07A(INPUT,OUTPUT);
      TYPE
        INT = 0..65535;
      VAR
        I:INTEGER;  J:INT;
      BEGIN
        READLN(I);
        J := I * 2;
        WRITELN('J = ', J);
      END.
      INPUT:          OUTPUT:
      2000              J = 4000
      -10000            J = -20000
      32000             J = -1536
      -32767            J =
      FIG USE07A PROGRAM

```

这个程序在编译和执行时都没有发生错误。但是，它的输出结果是不理想的。因为WRITE过程在输出整型量时，总是把一个字的最高位当作符号位处理。因此，当计算的结果在带符号整数范围内，它是按照带符号整数输出的。如输入 $I = 2000$ ， $I = -10000$ ，则分别输出 $J = 4000$ ， $J = -20000$ 。如果计算的结果超过带符号整数范围，它不进行溢出检查，所以不给出溢出错误。但WRITE过程仍把字的最高位作为符号位处理，不是作为32768。这样，当计算结果超过带符号整数范围时，输出结果相差65536。例如，输入 $I = 32000$ ，输出结果 J 不是64000，实际输出是 $64000 - 65536 = -1536$ 。再比如输入 I 的值为 -3267 ，输出结果 J 不为 -65534 ，实际输出是 $-65534 + 65536 = 2$ 。也就是说，当 J 的最高位为1时，WRITE过程给出的结果都不符合对 J 的说明。

因此，如果希望输出正确的结果，必须增加一个专用的输出过程。由于常用无符号整数表示内存地址，内存地址又多用八进制，所以我们的这个过程就实现用八进制形式输出 J 的值。请看程序FIG USE07B PROGRAM。

```

PROGRAM USE07B(INPUT, OUTPUT),
TYPE
  UNSIGNEDINT=0..65535,
VAR
  I, J:INTEGER;
  K :UNSIGNEDINT;
  PROCEDURE WRITEK,
  BEGIN
    FOR I:=1 TO 6 DO
      BEGIN
        CASE I OF
          1:
            BEGIN
              J:=K AND 100000B;
              IF J<>0
                THEN J:=1,
              K:=K AND 077777B
            END
          2: J:=(K AND 070000B) DIV 4096;
          3: J:=(K AND 007000B) DIV 512;
          4: J:=(K AND 000700B) DIV 64;
          5: J:=(K AND 000070B) DIV 8;
          6: J:=(K AND 000007B)
        END;
        WRITE(J:1)
      END;
    WRITELN('B')
  END;
BEGIN
  WRITE('INPUT I:'),

```

```

      READLN(I);
      IF I<0
      THEN WRITELN('INPUT ERROR')
      ELSE
        BEGIN
          K:=2*I;
          WRITE('OUTPUT K:');
          WRITEK
        END
      END.
INPUT I:-1
INPUT ERROR
INPUT I:0
OUTPUT K:000000B
INPUT I:1000
OUTPUT K:003720B
INPUT I:20000
OUTPUT K:116100B
INPUT I:32767
OUTPUT K:177776B
      FIG USE07B PROGRAM

```

改进以后的程序，输出的结果显然符合要求了。判 I 是否小于零，是因为 K 已被说明为 0..65535，用一个负的值去为它赋值是没有意义的事情。

如果想得到 J 的十进制形式的值，只要把 WRITEK 过程做一些修改就可以了。请看下一个程序 FIG USE07C PROGRAM。

```

      PROGRAM USE07C(INPUT, OUTPUT);
      TYPE
        UNSIGNEINT=0..65535;
      VAR
        I, J:INTEGER;
        K:UNSIGNEINT;
        B:BOOLEAN;
        A:ARRAY [1..5] OF 0..19;
        PROCEDURE WRITEK;
        BEGIN
          FOR I:=1 TO 5 DO A(I):=0;
          J:=K AND 100000B;
          IF J<>0
          THEN B:=TRUE
          ELSE B:=FALSE;
          K:=K AND 077777B;
          FOR I:=1 TO 5 DO
            BEGIN

```

```

        J := K MOD 10;
        K := K DIV 10;
        IF B
        THEN
            CASE 1 OF
                1: A [ I ] := A [ I ] + J + 8;
                2: A [ I ] := A [ I ] + J + 6;
                3: A [ I ] := A [ I ] + J + 7;
                4: A [ I ] := A [ I ] + J + 2;
                5: A [ I ] := A [ I ] + J + 3;
            END
            ELSE A [ I ] := A [ I ] + J;
            IF A [ I ] >= 10
            THEN
                BEGIN
                    A [ I ] := A [ I ] - 10;
                    A [ I + 1 ] := A [ I + 1 ] + 1;
                END
            END,
        FOR I := 5 DOWNT0 1 DO WRITE(A [ I ] :1)
        WRITELN
        END,
    BEGIN
        WRITE('INPUT I:'); READLN(I);
        IF I < 0
        THEN WRITELN('INPUT ERROR')
        ELSE
            BEGIN
                K := I * 2;
                WRITE('OUTPUT K:');
                WRITEK
            END
        END
    END
    INPUT I: -1
    INPUT ERROR
    INPUT I: 0
    OUTPUT K: 00000
    INPUT I: 1000
    OUTPUT K: 02000
    INPUT I: 20000
    OUTPUT K: 40000
    INPUT I: 32767
    OUTPUT K: 65534
    FIG USE0C PROGRAM

```

这个程序不难懂。在 WRITEK 过程中，首先把无符号整数 K 分解成两个部分，先看存放 K 的那个字的最高位是否为 1，如果是，就先把 K 分解为 32768 与 $(K - 32768)$ 这样两部分，然后把这两个部分逐位相加，结果存放在数组 A 中。输出时沿从高位向低位的方向逐位输出，就得到了正确的输出结果。这里要说明两点：

第一，在逐位相加时沿从低位向高位的方向逐位进行，但当本位结果大于或等于 10 时，本位应减 10，高位加 1，实现应有的进位操作；

第二，输出时沿从高位向低位的方向逐位进行，而且每位都用场宽 1。

本章小结

枚举类型和子界类型都是简单的数据类型，它们由用户根据需要确定。

枚举类型用于解决一些特殊的问题。它的数据都是用户定义的标识符，只有在用户定义之后才能使用。数据的名称必须符合用户标识符的规定。一般情况下，不能通过读写语句对这类数据进行输入或输出，只能间接地通过转换过程象征性地输入和输出。枚举型数据常常作为集合中的元素，作为循环控制变量，作为 CASE 语句中的情况标号，作为数组的下标等使用。

子界类型用于解决一些特定的问题。本质上是整数型、字符型和枚举型的局部。子界型数据的特点取决于其常量类型的特点。子界变量的说明与使用都很方便。用子界型数据书写的程序便于阅读，往往还有实现自动越界查错，可以节省内存空间等好处。数组的下标，用的就是子界类型数据。

本章习题

1. 为什么要使用枚举类型数据？枚举类型数据有什么特点？在使用中要注意哪些问题？枚举类型变量不能直接通过读写语句输入输出，在使用中如何解决这个问题？

2. 例 6-5 所示的程序 FIG USE05 PROGRAM 并不是一个最简化的程序，请你简化之，并从中体会枚举类型数据的使用要领。

3. 使用子界类型数据有什么好处？有哪些地方经常使用这种数据类型？为什么子界的上下界必须是属于同一类型的有序的数据类型？哪些数据类型可以采用？实数类型行吗？为什么？

第七章 构造型数据类型

第二章介绍的四种标准数据类型是 PASCAL 语言定义的。第六章介绍的两种数据类型是用户根据需要自己定义的。它们都属于简单的数据类型。简单数据类型直接由一系列元素组成，可以直接通过这类变量名来访问它的一个元素。

本章和下一章要讨论复杂数据类型——构造型数据类型。构造型数据是按照一定的规则组成的元素类型的数据。元素类型又叫基类型，它给出的是构造型数据本身的类型，既可以是简单数据类型，又可以是构造型数据类型，一定的规则体现为构造方法，它跟具体的类型有关。在一般情况下，如果要访问某一个元素，单纯用一个变量名称是不够的。

这一章介绍集合类型、数组类型和记录类型。它们提供了比较灵活的方法，使程序员可以在自己的程序中，方便地组织和使用自己的数据。

下一章单独介绍文件类型。通过它，程序员可以通过外存贮器来组织和使用自己的数据。

§1 集合类型

如果把某些具有共同特性（共性）而又能互相区别（个性）的对象聚集在一起，那么这样形成的整体叫集合（SET）。一个集合是同类型对象的一个汇集。集合中既互相联系又相互区别的对象叫这种集合的元素。没有任何元素的集合叫空集合。

例如，所有的大写英文字母是一个集合，它包括26个元素：A，B，……Z。其共性在于都是英文字母，其个性在于各自形状不同，读音不同。

在 PASCAL 语言中，集合类型定义为某种数据类型的值的集合，这个集合是它的元素类型的势集（POWERSET），也就是说，集合类型定义为基类型的值的所有子集（包括空集在内）的集合。其中元素类型必须是简单类型，包括枚举类型与子界类型。

集合是 PASCAL 语言中的构造型数据之一。它与我们后面将要介绍的其他构造型数据的相同点，就是它们都是收集起来的各自的一批基类型数据，但有明显的区别。后面介绍的三种构造型数据（数组、记录和文件），通常是把它们每一个元素作为一个独立的数据使用，而集合类型数据却往往把一个集合变量看作为一个完整的单一的数据使用。

为了给出一个集合的子集，我们总是用方括号〔 〕把给定的元素括起来，每个元素之间用逗号隔开，也可以用子界形式给出一批元素。

集合类型的定义格式如图7.1-1所示。

其中：

SET 和 OF 是集合类型的标志，都是保留关键字；

类型名由用户自己定义，必须符合标识符的规定；

元素类型必须是某些简单类型，包括枚举类型和子界类型。

例7-1 用枚举类型数据作为元素类型，定义一个集合类型，并进行简单的运算。

程序如 FIG SET01 PROGRAM 所示。

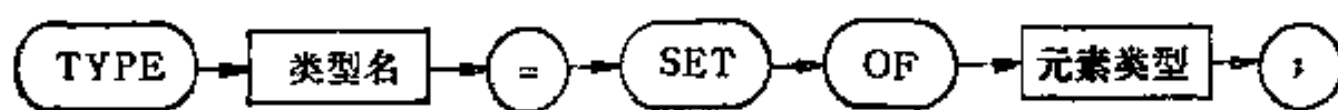


图 7.1-1 集合类型的格式

```

PROGRAM SET01 (INPUT, OUTPUT);
TYPE
  M=(A1, B2, C3, D4, E5, F6);
  S=SET OF M;
VAR
  X1,    X2    :M;
  Y1, Y2, Y3, Y4 :S;
BEGIN
  X1:=A1;      X2:=E5;
  Y1:=[A1] ;   Y2:=[B2, C3, D4] ;
  Y3:=[A1, B2, C3, D4, E5, F6] ;
  Y4:= [ ]
END.
  FIG SET01 PROGRAM

```

在这个程序中:

M是枚举类型标识符, 包含六个常量: A1, B2, C3, D4, E5, F6;

S是集合类型标识符, 元素类型是M。共有六个元素。

X1和 X2是枚举类型M的变量, 它只能是六个常量中的某一个常量。如 X1:=A1; X2:=E5; Y1到 Y4是集合类型S的变量, 它可以包含六个元素中的任意一个 (如 Y1:=[A1]) 也可以包含六个元素中的几个元素, 甚至全部元素。如 Y3:=[A1, B2, C3, D4, E5, F6], 集合也可以不包含任何元素, 形成空集合。如 Y4:= [] ;

从这里可以看出, 枚举类型可以作为集合类型的基类型。

例7-2 用子界类型作为元素类型, 定义一个集合类型, 并进行简单的运算。

程序如 FIG SET02 PROGRAM 所示。

```

PROGRAM SET02 (INPUT, OUTPUT);
TYPE
  M=1..20;
  S=SET OF M;
VAR
  X1,    X2    :M;
  Y1, Y2, Y3, Y4 :S;
BEGIN
  X1:=5;      X2:=15;
  Y1:= [1] ;   Y2:= [1, 3, 5] ;
  Y3:= [10..20] ;
  Y4:= [ ]
END.
  FIG SET02 PROGRAM

```


在这个程序中:

M是子界类型标识符, 包括20个整数。即1到20。

S是集合类型标识符, 其元素类型为M, 共有20个元素。

程序执行部分的基本语句与 SET01 PROGRAM 相同。突出的一点是: 如果集合变量中的元素是连续的, 也可以用类似于子界的方法来表示。例如, $Y3 := [10..20]$; 表示 Y3 中有11个元素, 从10到20。它与下列语句功能相同:

$Y3 := [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20];$

与枚举类型和子界类型相类似, 可以把集合的类型定义和变量说明合并在一起。

例7-3 定义元素类型为枚举类型, 子界类型和字符类型的集合, 进行简单的赋值运算和关系运算, 并输出比较的结果。

分析:

先定义一个类型 CHARSET, 它是字符类型的集合。然后对变量 Z1 和 Z2 加以说明, 表示 Z1 和 Z2 属于 CHARSET。

程序如 FIG SET03 PROGRAM 所示。

```
PROGRAM SET03 (OUTPUT);
TYPE
  CHARSET=SET OF CHAR;
VAR
  X, X1, X2:SET OF (A, B, C, D, E, F);
  Y1, Y2:SET OF 1..20;
  Z1, Z2:CHARSET;
BEGIN
  X:= (A, D, B, C);
  X1:=X;
  X2:= (A..D);
  WRITELN('X1=X2 IS', X1=X2);
  Y1:= (1, 2, 5, 15, 16, 17, 18, 3, 4);
  Y2:= (1..5, 15..18);
  WRITELN('Y1=Y2 IS', Y1=Y2);
  Z1 := ('0'..'9', 'A'..'Z')
END.
OUTPUT:
X1=X2 IS TRUE
Y1=Y2 IS TRUE
NOTE:
THE Z2:= (' '..'~')
IS 'BAD SET ELEMENT'
FIG SET03 PROGRAM
```

在这个程序中:

已把类型定义和变量说明合并在一起。

可以对集合类型变量直接进行赋值运算, 将某几个元素赋值给变量。例如, 语句:

$X := (A, D, B, C);$

如果两个变量都属于同一种集合类型，那么可以将一个变量的值赋给另一个变量，例如语句 $X1:=X$;

如果一个集合中的元素是连续的枚举类型的数值，那么，同样可以用类似于子界的方法来表示。例如，语句 $X2:= [A..D]$ 。又如，数字字符和字母字符的集合可以用如下语句表示：

$$Z1:= ['0'..'9', 'A'..'Z'];$$

一个有N个元素的集合类型，可以组成 $N!$ 种集合。

为了高效率地进行集合运算，PASCAL语言的报告建议限制集合类型中元素的个数。

OMSI PASCAL-1语言规定：集合类型中元素的个数不能超过64个。因此，如果一个集合类型的元素类型是字符类型，那么这种集合类型最多也只有64个元素，而不是ASCII码表中所有的可打印的95个元素。所以它仅仅是字符的子集。

请注意类型定义：

TYPE

CHARSET=SET OF CHAR;

完全等同于类型定义：

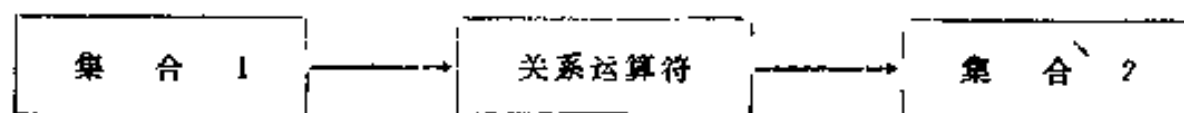
TYPE

CHARSET=SET OF ' '..'-';

其中空格的字符码为32，'-'的字符码为95，正好是64个元素。

如果某个集合超过了64个元素，那么在编译时会指出“集合元素有错”(BAD SET ELEMENT)。例如，语句 $Z2:= ['_'\dots'\sim']$ 是非法的。其中字符码大于95的字符不在这个集合类型之中。这就是这个程序后面的注解 (NOTE) 所表示的意思。

集合不仅可以进行赋值运算，还可以进行关系运算。两个集合通过关系运算符进行比较，结果为布尔型。集合类型数据进行关系运算的格式如下：



集合关系运算符一般有四种，即：

= 表示两个集合相等；

<> 表示两个集合不等；

>= 表示前者蕴含后者；

<= 表示前者蕴含于后者。

如果集合1和集合2中的元素个数相等，内容相同，不管排列顺序如何，都叫这两个集合相等。反之，则为不相等。例如：

$[A, B, C] = [B, A, C]$ 为TRUE;

$[A, B, C] <> [B, C, D]$ 为TRUE;

如果集合2的所有元素在集合1中都可以找到，那么叫集合1蕴含集合2。反之，则叫集合1蕴含于集合2。例如：

$[A, B, C] >= [A]$ 为TRUE;

$[A, B, C] <= [A, B, C, D]$ 为TRUE;

请注意，关系运算符“>”和“<”不能运用于集合的比较。由于把集合作为一个完整

的单一数据使用，所以序数函数 ORD 等也不能用于集合的运算。这是与枚举型变量的又一重要区别。

IN，是集合的特殊的关系运算，就是判断某些元素是否属于这个集合。如果属于则为真，否则为假。例如：

A IN {A, B, C} 为TRUE;

A IN {B, C, D} 为FALSE;

它们表示元素A在集合 {A, B, C} 中，不在集合 {B, C, D} 中。

IN 的使用大大缩短了运算时间，同时便于书写和阅读，这是集合突出的优点。

例如，如果希望将输入的数字字符进行计数，其余的不必计数，一般情况下用如下语句：

```
READ(CH);
IF (CH='1') OR (CH='2') OR (CH='3') OR (CH='4')
    OR (CH='5') OR (CH='6') OR (CH='7')
    OR (CH='8') OR (CH='9') OR (CH='0')
    THEN I:=I+1;
```

这样表示虽然正确，但是语句太长，执行起来太慢。也可以写成：

```
IF (CH>='0') AND (CH<='9')
    THEN I:=I+1;
```

但用 IN 关系运算更方便。例如：

```
READ(CH);
IF CH IN ('0'..'9')
    THEN I:=I+1;
```

这样，程序十分清晰，运行速度特别快。

例7-4 输入一系列字符，将数字字符，字母字符和其它字符分别计数，并输出计数的结果。输入字符 '?' 后不再计数。

程序如FIG SET04 PROGRAM所示。

```
PROGRAM SET04(INPUT, OUTPUT);
VAR
  ID, IL, IO: INTEGER;
  CH : CHAR;
  LETTER : SET OF 'A'..'Z';
  DIGIT : SET OF '0'..'9';
BEGIN
  LETTER:= ('A'..'Z');
  DIGIT:= ('0'..'9');
  IL:=0; ID:=0; IO:=0;
  READLN;
  REPEAT
    READ(CH);
    IF CH IN LETTER
```

```

        THEN IL:=IL+1
        ELSE
        IF CH IN DIGIT
        THEN ID:=ID+1
        ELSE IO:=IO+1
    UNTIL CH='?';
    Writeln('LETTER:',IL:5);
    Writeln(' DIGIT  ',ID:5);
    Writeln(' OTHER:',IO:5)
END.
INPUT:
012345ABCD($*); [ ]
OUTPUT:
LETTER:    4
  DIGIT:    6
  OTHER:    5
      FIG SET04  PROGRAM

```

集合不能进行与、或、非的逻辑运算，因为集合类型数据不是布尔量。但是，它们可以进行三种特殊的逻辑运算，通常叫做集合运算。

(一) 并 (UNION) 运算:

设M, N为两个集合。由属于M或者属于N的全部元素，即M集合中的元素，加上N集合中的与M集合不同的那些元素组成的集合，称为集合M和N的并。并运算符为“+”，记作M+N。

例如

$\{A, B, C\} + \{D\}$ 为 $\{A, B, C, D\}$;
 $\{A, B, C\} + \{B\}$ 为 $\{A, B, C\}$;
 $\{A, B, C\} + \{B, C, D\}$ 为 $\{A, B, C, D\}$;

由于在一个集合中，各个元素既互相联系（类型相同），又相互区别，因此，两个集合的共同元素在新集合中只能出现一次。不能写成 $\{A, B, C, B, C, D\}$ 。

(二) 交 (INTERSECTION) 运算:

设M, N为两个集合。由既属于M又属于N的所有元素，即两个集合中都具有的那些元素组成的集合，称为集合M和N的交。交运算符为“•”，记作M•N。例如：

$\{A, B, C\} \bullet \{B\}$ 为 $\{B\}$;
 $\{A, B, C\} \bullet \{B, C, D\}$ 为 $\{B, C\}$;
 $\{A, B, C\} \bullet \{D\}$ 为 $\{\}$;

如果两个集合没有共同的元素，则它们的交为空集合。

(三) 差 (DIFFERENCE) 运算:

设M, N为两个集合，由属于M而不属于N的所有元素，即M集合的元素除去N集合中那些与M集合相同的元素后的所有元素组成的集合，称为集合M和N的差。差运算符为“-”，记作M-N。

例如：

$\{A, B, C\} - \{B\}$ 为 $\{A, C\}$;
 $\{A, B, C\} - \{D\}$ 为 $\{A, B, C\}$;
 $\{A, B, C\} - \{B, C, D\}$ 为 $\{A\}$;

在运算中, 不必考虑那些仅仅属于N而不属于M的元素, 如D。

为了直观一些, 我们用图示法来表示这三种运算。其中圆圈M的内部表示集合M, 圆圈N的内部表示集合N, 斜线部分表示集合运算的结果。

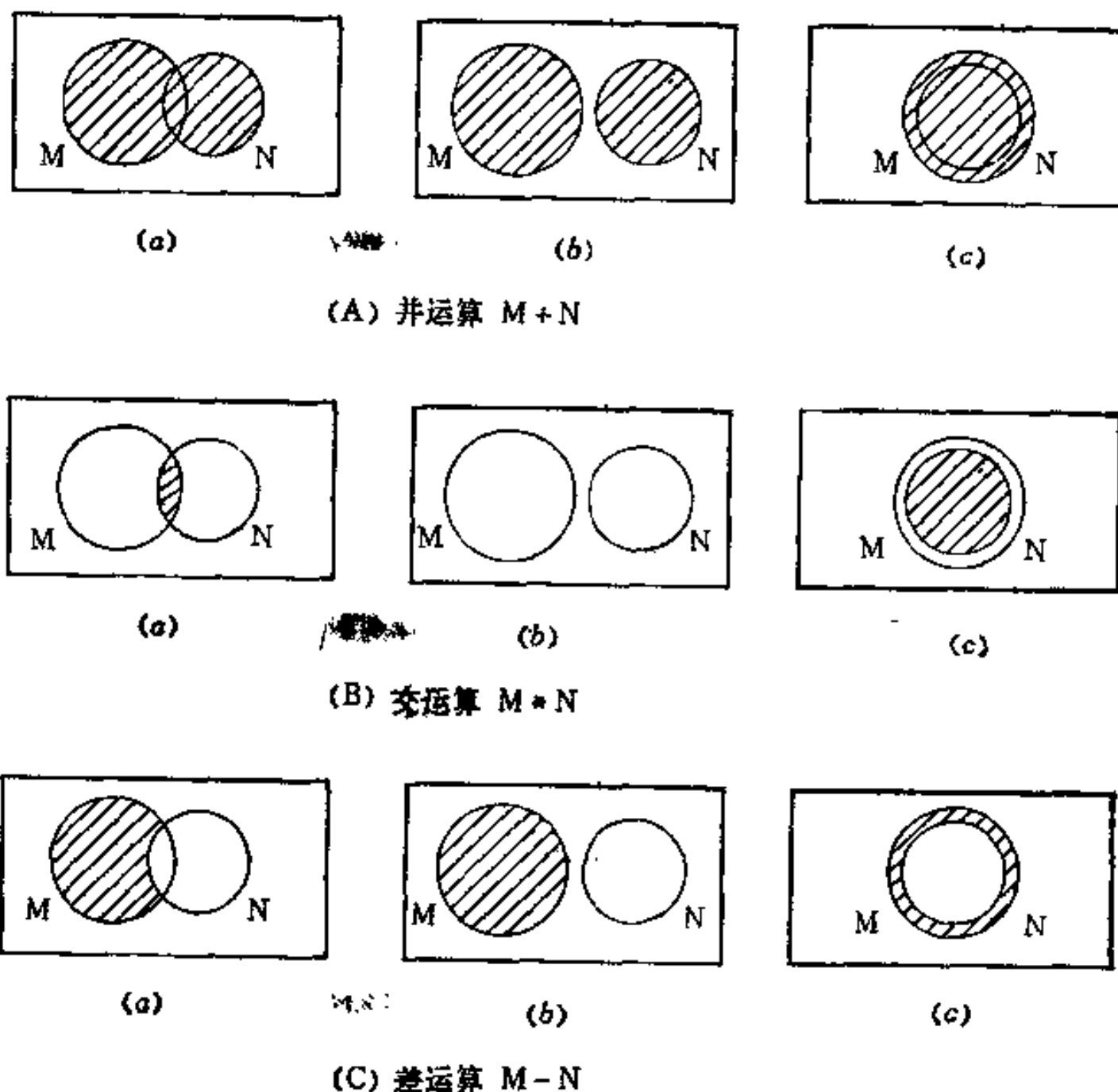


图 7.1-2 集合运算的示意图

由于这三种集合运算的存在, 赋值语句的应用又可以进一步发展。赋值语句的右边可以是集合表达式。例如:

$X_1 := \{A\}$;

$X_2 := \{B, C\} + X_1$;

运行之后, X_2 中就会有三个元素: A, B和C。

又如:

$I := 5$;

$Y := \{1, 2, 3, 4\} + \{I\}$;

运行之后, Y中就会有五个元素: 1, 2, 3, 4, 5。

必须指出, 如果用这样的语句:

$Y := \{1, 2, 3, 4\} + I$;

这是不行的。因为I是整数类型变量, 如果要变成集合中的元素, 则必须加上方括号

[]。

集合中的元素不能作为常量加以定义。集合类型变量不能进行算术运算。即不允许直接用读语句输入集合，也不允许直接用写语句输出集合。如果希望输入或输出，必须通过间接的方法进行。

例7-5 建立两个集合，然后对它们进行并运算、差运算和交运算，最后间接输出运算的结果。

程序如FIG SET05 PROGRAM所示

```
PROGRAM SET05 (INPUT,OUTPUT),
TYPE
  M=(A,B,C,D,E,F),
  N=1..20,
  S=SET OF M,
  R=SET OF N,
VAR
  X1,X2,Z1,Z2,Z3:S,
  C1 C2 :BOOLEAN,
  J :M,
PROCEDURE SETOUT(Z:S),
VAR
  CH:CHAR, K:M,
BEGIN
  WRITE('[');
  FOR K:=A TO F DO
    IF K IN Z
    THEN
      BEGIN
        CH:=CHR(ORD(K)+65),
        WRITE(CH:2)
      END,
  WRITELN('] ')
END;
BEGIN      (* MAIN PROGRAM *)
  (* PART 1..... CREATE SET *)
  X1:=(A,B,C)
  X2:= ( ) ;
  FOR I:=B TO D DO
    X2:=X2+(I) ;
  (* PART 2..... SET OPERATE *)
  Z1:=X1+X2,
  Z2:=X1-X2,
  Z3:=X1•X2,
  C1:=X1<=X2,
  C2:=D IN Z1,
```

```

(• PART 3..... OUTPUT SET •)
WRITE('X1='); SETOUT(X1);
WRITE('X2='); SETOUT(X2);
WRITE('Z1='); SETOUT(Z1);
WRITE('Z2='); SETOUT(Z2);
WRITE('Z3='); SETOUT(Z3);
WRITELN('C1= ', C1);
WRITELN('C2= ', C2)
END.

```

```

OUTPUT:
X1= {A B C}
X2= {B C D}
Z1= {A B C D}
Z2= {A}
Z3= {B C}
C1=FALSE
C2= TRUE

```

FIG SET05 PROGRAM

从这个程序可以看出以下三点:

1. 集合的建立:

集合的建立可以通过赋值语句直接实现, 如 X1, 也可以先初始化一个集合, 然后通过并运算向集合中逐步加入各个元素, 形成完整的集合 如 X2.

2. 集合的运算

集合的基本运算是赋值运算 关系运算和特殊的逻辑运算. 赋值运算如 X1, 关系运算如 C1和 C2, 特殊的逻辑运算如 Z1, Z2和 Z3.

3. 集合的输出:

集合的输出必须间接地转换, 不能直接通过写语句输出某个集合变量. 如果集合中的元素是数字或字母, 那么可以通过序数值的转换关系输出对应的字符, 但是, 它并不能代表集合, 只能作为用户检验程序的参考. 如过程 SETOUT.

注意的是, 枚举类型的序数值是从 0 开始的. 所以, 在转换时要加上相应的字符的序数值. 例如, 例题中字符 A, 要加入 A 的序数值65.

例7-6 根据用户的需要建立一个集合 然后从这个集合中清除部分元素, 最后输出集合中剩余的元素. 编制程序实现上述功能.

程序如FIG SET06 PROGRAM所示

```

PROGRAM SET06 (INPUT,OUTPUT),
TYPE
  M=1..20,
  S=SET OF M,
VAR
  X      Y1,Y2:S,
  I      :M,
  J      :INTEGER,

```

```

BEGIN      (•   MAIN PROGRAM   •)
  (•   PART 1..... INPUT   SET   •)
  X:= (1..20) ,      Y1:= ( ) ,
  READ(I),
  WHILE I IN X DO
    BEGIN
      Y1:=Y1+ (I) ,
      READ(I)
    END.
  (•   PART 2..... DELETE SET   •)
  Y2:=Y1,
  FOR J:= -10 TO 15 DO
    IF J IN Y2
      THEN Y2:=Y2- (J) ,
  (•   PART 3..... OUTPUT SET   •)
  WRITE('Y1= (');
  FOR I:=1 TO 20 DO
    IF I IN Y1
      THEN WRITE(I:4),
  WRITELN(' ] ');
  WRITE('Y2= (');
  FOR I:=1 TO 20 DO
    IF I IN Y2
      THEN WRITE(I:4),
  WRITELN(' ] ');
  END
INPUT:
  1   4   7  10  13  16
 17  18  19  20  25
OUTPUT:
Y1= (   1   4   7   10  13  16  17  18  19  20  )
Y2= (   16  17  18   19  20  )
INPUT:
  1   5   10  15  20 -25
OUTPUT:
Y1= (   1   5  10  15  20  )
Y2= (  20  )
INPUT:
 -10   0  15
OUTPUT:
Y1= (   )
Y2= (   )

```

FIG SET06 PROGRAM

这个程序分成三部分：

1. 集合的输入：

如果集合中的元素是整数或字符，那么可以通过读语句不断输入这些元素，然后通过并运算形成完整的集合。为了避免其它数据误入集合，必须首先判断输入的数据是否属于元素类型，然后再决定是否将它并入集合。因此，一般用当语句而不用直到语句。如程序第一部分。

2. 集合元素的取消：

集合元素的取消一般通过循环语句逐步实现，循环语句的成份语句是进行差运算。如果必要的话，可以将集合元素全部取消，形成空集合。如程序第二部分。

3. 集合的输出：

为了输出集合中的整数或字符元素，可以通过循环语句来实现。循环语句的成份语句是条件语句，如果这个元素在集合中则按规定输出，如程序第三部分。

思考题：

1. 上述程序为什么输入第三组数据时，Y1和Y2为〔 〕？

2. 如果将语句 $Y1 := Y1 + (I)$ 和 $Y2 := Y2 - (J)$ 修改成 $Y1 := Y1 + I$ 和 $Y2 := Y2 - J$ 行不行？为什么？

我们不准备详细说明一个集合在计算机中的表示方式，只是提醒读者注意，OMSI PASCAL-1语言规定每个集合最多用四个字长表示，即 $16 \times 4 = 64$ 位二进制数表示。每一位上的1代表一个对应元素在集合中，0则表示它不在集合中。至于代表的元素是整数、字符，还是别的什么，则由该集合的基类型决定。

集合类型是一种使用简便，节省内存而又运算速度快（因为它里面没有费时的算术运算，只有简单的几种特定的逻辑运算与判断）的数据类型，它很类似于PL/1语言中的位处理。在解决某些问题时，它能使我们的程序编写方便简明清晰，节省内存，又大大节省运行时间。具体例子可以在本章第§4看到，我们不在这里更多讨论。

§2. 数组类型

数组类型是一种常用的构造型数据类型。它由固定数量的同类元素组成。这就是说，一方面，它的元素的数量必须是确定的，不允许是变动的；另一方面，它的元素的类型必须是相同的，不允许是混合的，这就是数组类型的两个特点。

使用数组类型的最大好处是，可以让一批相同性质的数据共用一个变量名，而不必为每一个数据选定一个名字。不仅程序书写大为简便清晰，提高了程序的可读性，而且便于用重复语句简单处理这类数据。例如，我们可以用数组变量SINX来记忆 $0^\circ \sim 89^\circ$ 之间每隔一度的正弦值，用SCORE来表示一个学生所学过的30门课程中每门课程的分，用NAMES表示一个班45名学生的姓名等等。

当然，不能简单地用SINX来同时表示90个不等的数，我们还必须把90个数按一定规则组织起来，这正是构造类型数据的基本特点。最简便的方法，是把它们按照某种选定的序号（如 $0 \sim 89$ ）排列起来，这样我们就可以通过变量名字SINX与每一个正弦值的序号来访问这90个中的每一个数了。通常我们称这里的序号为下标，并把它写在方括号中，规定下标写在变量名之后，如图SINX(0)表示 0° 的正弦值，SINX(1)表示 1° 的正弦值，……SINX(89)

表示 89° 的正弦值，这样不仅看起来清楚，也更容易处理。

在具体介绍数组类型变量的说明与使用之前，先看一个具体例子：对从 0° 到 89° ，计算并输出每隔一度对应的正弦函数值。

用数组的办法，程序如下：

```
PROGRAM ARRAY00;
TYPE
  T=0..89;
  AR=ARRAY (T) OF REAL;
VAR
  I: INTEGER; R: REAL;
  SINX: AR;
BEGIN
  R:=3.14159/180;
  FOR I:=0 TO 89 DO
    BEGIN
      SINX(I):=SIN(I*R);
      WRITE(SINX(I):11:5);
      IF (I+1) MOD 5=0
        THEN WRITELN
      END
    END
  END.
```

这个程序非常简单，简单几行，就求出了 $0^\circ \sim 89^\circ$ 之间的每隔一度的正弦值，并把结果记入在数组变量SINX的90个元素中。

可以简单的解释几句。

在变量说明部分，用SINX:ARRAY(T) OF REAL定义了一个名字为SINX，由90个实数元素组成的数组。

在程序的执行部分，用 $R:=3.14159/180$ 求出角度一度所对应的弧度值，因为正弦函数的自变量必须用弧度作单位。循环语句的控制变量从0变到89，则SINX(I)也就跟着从SINX(0)变到SINX(89)，SIN(I*R)求出的正好是 0° 到 89° 之间每一度的正弦值。这里的赋值语句则完成把求出的结果记入SINX的90个元素中的每个相应元素的存贮单元中去。

现在再回过头来设想一下，如果在程序中不用数组变量SINX，而只用简单变量来表示这90个正弦值，就麻烦得多了。上述简单的程序可能要变成如下形式：

```
PROGRAM ARRAY01;
VAR
  R, SIN0, SIN1, SIN2, ... SIN89: REAL;
  I: INTEGER;
BEGIN
  R:=3.14159/180;
  SIN0:=SIN(0*R);      WRITELN(SIN0:11:5);
  SIN1:=SIN(1*R);      WRITELN(SIN1:11:5);
```

```
SIN 2 := SIN(2 * R);      WRITELN(SIN2:11:5);
```

```
SINX89: SIN(89 * R);      WRITELN(SIN89:11:5)
```

```
END.
```

上面这个程序，仅仅是个示意性的程序，实际上是不能运行的，如果要按实际运行要求，恐怕就要写好几页。

在这个程序中，为了记忆90个正弦值，就要说明90个实数型变量，求90个正弦值也要用90个赋值语句，输出90个正弦值，又要用90个写语句。因为每个正弦值的变量名不同，无法使用循环语句。

对比一下前后两个程序，可以清楚地看到使用数组类型数据的必要性。再设想一下，如果要把要处理的 $0^\circ \sim 89^\circ$ 变成 $0^\circ \sim 359^\circ$ ，对第一个程序来说，只要简单地把类型说明与循环语句中的89改成359就行了，其它可以丝毫不动，而后一个程序，还要在变量说明中再说明270个实型变量，并在程序中的执行部分增加270个赋值语句。这是实在无法接受的方案。

可能有些人立刻发现，对三角函数来说，实际上用不到求 $0^\circ \sim 359^\circ$ 每一度的函数值，对我们的例子只要求出 $0^\circ \sim 45^\circ$ 之间的值就行了，其余角度的值都可以通过简单运算得到。问题是，并不是所有问题都有这种关系，甚至于某些数据不是依照一定函数计算出来的。如一个学生30门课程中每门课程的分数就是这样，它可以是通过终端键盘打进去的，如果不用数组变量表示，则只能为每门课程给出变量名，并把它们作为 READ 过程的参数写在 READ 语句中。要求出30门课程的总分，就得用30个变量相加的办法来实现，其烦人程度可想而知。

简言之，数组类型是一种最常用的构造数据类型，必须很好地掌握。

下面详细地介绍数组类型变量的说明与使用。

数组类型定义的一般格式如图7-2-1所示：

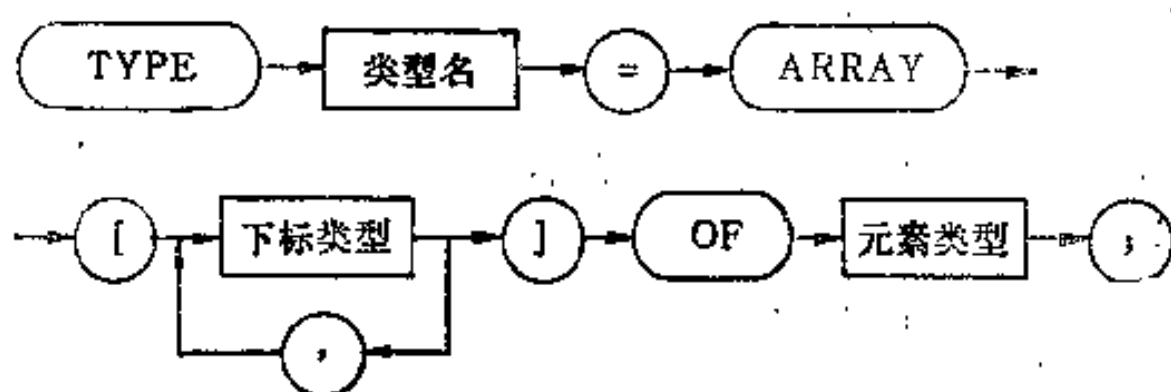


图 7.2-1 数组类型定义的格式

其中：

ARRAY和OF是数组类型的标志；它们都是PASCAL语言的保留关键字。类型名由用户自己定义，它必须符合标识符的规定；

下标类型一般为简单类型，包括整数类型，字符类型，枚举类型和子界类型。实数类型不行。它给出的值是数组中每个元素的下标的值。

元素类型又叫基类型，它给出的是数组变量的元素类型。它可以是任何数据类型，甚至是构造型数据类型。但是要注意，在同一个数组中，所有元素必须是同一数据类型。

例如，在上面的程序 ARRY00 例题中，AR 是一种数组类型，其下标类型为子界类型 0..89，其元素类型为实数类型。

一个变量可以是某一种数组类型的变量，它具有这种数组类型数据的特性。一般定义如图 7.2-2 所示。

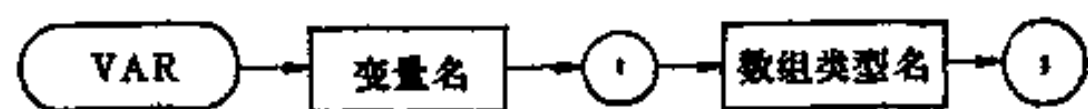


图 7.2-2 数组类型变量说明的格式

例如：

VAR

SINX:AR;

表示变量 SINX 属于 AR 数组

类型。

同样，数组类型说明和变量说明可以合并在一起 这样可以减少用户标识符。例如，

VAR

SINX:ARRAY [0..89] OF REAL;

为了确切地表达一个数组型变量的元素，一般必须弄清三个问题：

1. 这个数组叫什么名称？
2. 它是数组类型中的第几项？
3. 数组元素属于哪一种类型？

对于第 3 个问题，在类型说明时已经解决。因此，表达一个数组类型变量的格式如下，

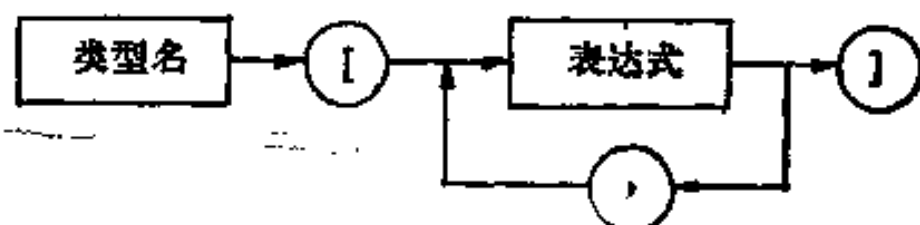


图 7.2-3 数组的项

其中表达式表示的是数组的下标，即数组的第几项。例如：

SINX(0)，表示数组 SINX 的第 0 项；

SINX(10)，表示数组 SINX 的第 10 项；

SINX(I+1)，表示数组 SINX 的第 I+1 项；

它们都是实数类型数据。

必须指出，下面的概念要记清，不能混淆；

数组类型标识符表示一类数据类型；

数组类型变量名表示一个数组类型的变量；

SINX(4)表示一个具体的数组元素的值。

其次，下标类型与下标的概念也有区别，下标一般指的是下标类型中的某一具体的值。

一定要弄清楚数组的下标类型与元素类型的区别以及它们各自的作用。

在使用数组变量时，PASCAL 语言一般不允许把一个整个的数组作为一个单个数据进行访问，每次只允许通过数组变量名与相应下标访问数组变量的一个元素。如果数组元素已经是简单变量，则访问与使用每个元素的规则与访问相同类型的简单类型数据是极为类似的，所差的仅是访问数组元素时用的是数组变量与该元素的相应下标，访问简单类型数据用的则是相应变量名。请看下面一个例子。

例 7-7 定义两个数组类型，其下标类型分别为子界类型和枚举类型，其元素类型分别

为整数类型和实数类型。建立一个包含十项由整数元素构成的数组，然后将这数组中间的六项内容依次传送给另一个包含六项由实数元素构成的数组，最后输出后一个实数数组。

程序如FIG ARY01 PROGRAM所示。

```

PROGRAM ARY01 (INPUT, OUTPUT),
TYPE
  M=1..10,
  N= (A, B, C, D, E, F),
  P=ARRAY (M) OF INTEGER,
  Q=ARRAY (N) OF REAL,
VAR
  I:M,      J:N,
  X:P,      Y:Q,
BEGIN
  (• PART 1.....INPUT ARRAY •)
  X (2) :=8,
  Y (E) :=3.5,
  FOR I:=1 TO 10 DO READ (X (I)),
  (• PART 2.....ARRAY OPERATE •)
  I:=2,
  FOR J:=A TO F DO
  BEGIN
    I:=I+1,
    Y (J) :=X (I)
  END,
  (• PART 3.....OUTPUT ARRAY •)
  I:=0,
  WRITELN ('SIX VALUES OF THE ARRAY Y ARE:'),
  FOR J:=A TO F DO
  BEGIN
    WRITE (Y (J) , ' ',6),
    I:=I+1,
    IF I MOD 2=0
    THEN WRITELN
  END
  END.
INPUT,
11 22 333 4455 6666 16789 32767 110 99 7765
OUTPUT,
SIX VALUES OF THE ARRAY Y ARE,
3.330000E+02      4.455000E+03
6.666000E+03      1.678900E+04
3.276700E+04      1.100000E+02
EIG ARY01 PROGRAM

```

在这个程序的说明部分，没有任何标准数据类型，全是用户定义的数据类型。P为10个整数的数组类型。Q为6个实数的数组类型。变量X、Y分别为P类型和Q类型的变量。类型定义和变量说明没有合并。

在这个程序的执行部分，由三个部分组成，

1. 数组值的输入

可以通过赋值语句对数组的逐个元素进行赋值。如上述例题的程序中的 $X(2) := 8$ ， $Y(E) := 3.5$ ；也可以通过读语句输入，如程序中 $\text{FOR } I := 1 \text{ TO } 10 \text{ DO READ } (X(I))$ ；通过循环语句对一个完整的数组的每一个元素输入相应的数据。

2. 数组的运算

将数组X的第三项到第八项依次赋值给数组Y的六项。这里将整数作为实数使用，这是允许的。赋值的过程是通过循环语句实现的。循环语句的成分语句中，其一是改变数组X的下标，其二是进行数据的传送。

3. 数组值的输出

可以通过写语句对数组的值进行输出。一个完整的数组的全部的输出，必须通过循环语句来实现。输出的格式由用户根据需要确定。在本题中，数组Y有六项即有六个实数，每一行输出两个实数。

定义数组类型时，必须严格按照其规定的格式书写，注意它与其它算法语言的区别。如果概念模糊或粗心大意，类型定义时很容易出错。

例如，将数组类型Q定义如下：

TYPE

Q=ARRAY (A,B,C,D,E,F) OF REAL;

粗看起来，这个类型定义似乎没有什么错误。但是在编译这个程序时，马上进入陷阱 (TRAP)，并不指出任何具体的错误。

思考题：

这个类型定义究竟错在哪儿？答案在下面。

请看下一个例题。

例7-8 数组类型定义，变量说明及进行运算时容易出现各种各样的错误。为了巩固基本概念，编制一个包含九个错误的程序。

程序如FIG ARY02 PROGRAM所示。

```
PROGRAM ARY02 (INPUT, OUTPUT),
TYPE
  M=1..10;
  N=(A, B, C, D, E, F);
VAR
  I, INTEGER,
  J, 'A'..'F',
  X, X1, X2, ARRAY (1..5) OF INTEGER,
  Y      :ARRAY (M ) OF REAL,
  Z      :ARRAY (N ) OF CHAR,
  W      :ARRAY (J ) OF BOOLEAN,
  (* BAD TYPE SPECIFICATION *)
```

```

    (• EXPECTED ' ] ' MISSING •)
    (• EXPECTED 'OF' MISSING •)
    (• BAD TYPE SPECIFICATION •)
    (• ARRAY INDEX TYPE ERROR •)
BEGIN
    X1:=X2,
    Y:=X,
    (• INCOMPATIBLE TYPE •)
    X(2):=ROUND(Y(3)),
    Y(1..5):=X(1..5);
    (• EXPECTED ' ] ' MISSING •)
    (• BAD EXPRESSION •)
    (• INVALID SYMBOL •)
    FOR I:=1 TO 5 DO Y(I):=X(I)
END.
(• THERE ARE 9 ERRORS IN THE PROGRAM •)
FIG ARY02 PROGRAM

```

在这个程序中，我们合并了部分类型说明和变量说明。在变量W的说明下面，指出了五个错误，其实关键的错误只有一个：“数组下标类型错误”(ARRAY INDEX TYPE ERROR)。这个数组下标类型应该为一个简单数据类型，不能是简单数据类型的变量。这儿，J是子界类型变量，不允许作为下标类型。如果将J定义为一个类型，例如：

```

TYPE
    J='A'..'F';

```

那么上述五个错误都没有了。

程序执行部分共有五条语句：

语句1为X1:=X2；它表示：如果两个数组类型变量属于同一数组类型，那么，通过一个赋值语句就可将一个数组各元素的值赋给另一个数组对应的各元素。实质上，语句X1:=X2的功能与下列五个赋值语句的功能相同：

```

X1(1):=X2(1);      X1(2):=X2(2);
X1(3):=X2(3);      X1(4):=X2(4);
X1(5):=X2(5);

```

如果数组的项越多，则这条语句的优越性越强。

语句2为Y:=X；尽管它与语句1形式相似，但是由于Y和X为不同类型的数组，表现为下标类型和元素类型都不相同。因此，不允许作为整体直接赋值。否则，编译时指出错误类型为“类型不相容”(INCOMPATIBLE TYPE)。

语句3为X(2):=ROUND(Y(3))；它表示将数组Y的第3项(实数值)截尾后作为数组X的第2项(整数值)，从而完成实数向整数的转换。

语句4为Y(1..5):=X(1..5)；其目的是将数组X的各项数值赋给数组Y的前五项。尽管将整数作为实数使用是允许的。但是，这种表示方法是错误的。三个错误的核心是，“表达式有错”(BAD EXPRESSION)。

语句5为FOR I:=1 TO 5 DO Y(I):=X(I)；它完成了语句4没有完成的任务。将

数组X的各项值(共5项)对应地赋值给数组Y的前5项。

思考题答案:

类型定义

TYPE

Q=ARRAY (A, B, C, D, E, F) OF REAL;

的错误在于将一对方括号[]写成一对圆括号()。许多初学者曾为此付出了时间和精力。

矩阵运算是经常遇到的问题。

学了数组,我们找到了矩阵运算的比较简单的方法。大家知道,数组的元素类型允许为任何类型,那么可以是数组吗?回答是肯定的。数组型数组就是矩阵(MATRIX)。

例如:

TYPE

N=ARRAY[1..5] OF REAL;

M=ARRAY[1..10] OF N;

VAR

X, Y, Z : N;

A, B, C : M;

这些说明表示, N是由5个实数组成的数组类型。X, Y, Z为N型的变量。M是由10个N型数据组成的数组,即一个矩阵型。A, B, C为矩阵型M的变量。矩阵型的定义可以简化一下,变为:

TYPE

M=ARRAY[1..10, 1..5] OF REAL;

VAR

A, B, C, M;

或者

VAR

A, B, C, ARRAY [1..10, 1..5] OF REAL;

在这里, M的下标类型有两个,我们称为二维数组。推而广之,有n个下标类型的数组就叫n维数组。M型数据有10行,5列,共50个元素,其中任何一个元素可以通过A(I, J)来表示。A(I, J)表示矩阵中第I行第J列的那个元素。例如,二维数组M可以想象为下述存贮方式:

A(1, 1) A(1, 2) A(1, 3) A(1, 4) A(1, 5)

A(2, 1) A(2, 2) A(2, 3) A(2, 4) A(2, 5)

.....

A(10, 1) A(10, 2) A(10, 3) A(10, 4) A(10, 5)

当然,这并不是说,它们在计算机存储器中就是这样整齐地排列的。事实上,为了节省内存空间,它们在内存中仍然是按照一维数组一行接一行顺序排列的。但是, PASCAL的编译程序能够圆满地解决这个问题。三维数组或多维数组也是如此。

在线性代数中,矩阵的运算,尤其是矩阵的乘法运算占很大比重。下面我们介绍用PASCAL语言进行矩阵乘法的方法。

例7-9 请编制程序完成下述功能:建立矩阵A和B,矩阵A和B相乘得矩阵C,输出

矩阵C。

分析：

矩阵相乘要求矩阵A的列数和矩阵B的行数相等。否则无法计算。我们假定矩阵A为四行三列，矩阵B为三行两列，那么，根据矩阵相乘的规则，矩阵C应该为四行两列。

程序如FIG ARY03 PROGRAM所示。

```
PROGRAM ARY03 (INPUT, OUTPUT),
CONST
  M=4,    P=3,    N=2,
VAR
  I:1..M,  J:1..N,  K:1..P,
  A:ARRAY (1..M, 1..P) OF INTEGER,
  B:ARRAY (1..P, 1..N) OF INTEGER,
  C:ARRAY (1..M, 1..N) OF INTEGER,
BEGIN
  FOR I:=1 TO M DO
    BEGIN
      FOR K:=1 TO P DO READ (A (I, K) ),
        READLN ;
      END;
    READLN;
    FOR K:=1 TO P DO
      BEGIN
        FOR J:=1 TO N DO READ (B (K, J) ),
          READLN ;
        END;
        FOR I:=1 TO M DO
          BEGIN
            FOR J:=1 TO N DO
              BEGIN
                C (I, J) :=0;
                C (I, J) :=C (I, J) +A (I, K) *B (K, J) ,
                FOR K:=1 TO P DO
                  WRITE(C (I, J) :6)
                END;
              END;
            WRITELN
          END;
        END;
      INPUT,
      1      2      3
      - 2    0      2
      1      0      1
      - 1    2      - 3
```

```

- 1  3
- 2  2
  2  1

```

OUTPUT,

```

  1  10
  6  -4
  1   4
-9  -2

```

FIG ARY03 PROGRAM

在这个程序中:

1. 矩阵A或矩阵B的建立都是通过两重循环执行读语句 (READ (A(I, K)) 或 READ (B(K, J)) 来实现的。READLN语句的使用是为了输入数据时格式整齐, 犹如一个矩阵形式;

2. 矩阵C的建立和输出是通过三重循环实现的。最核心的是赋值语句;

$C(I, J) := C(I, J) + A(I, K) * B(K, J);$

表示矩阵A的第I行, 第K列的那个元素与矩阵B的第K行、第J列的那个元素相乘得到矩阵C的第I行、第J列的元素值。

WRITELN 语句的使用是为了输出数据时格式整齐, 好似一个矩阵形式。

字符数组是经常使用的一种数组, 由N个字符组成的字符串定义如下:

TYPE

STRING=ARRAY(1..N) OF CHAR;

其中:

N必须大于1;

N必须是已知数。

这是因为PASCAL语言没有动态数组。串的长度必须在程序设计过程中确定, 不允许在程序执行过程中通过读语句输入。

如果N=10, PASCAL语言把它定义为一种标准的数组类型。取名为ALFA。即:

TYPE

ALFA=ARRAY(1..10) OF CHAR;

其中:

ALFA为标准关键字, 用户可以直接使用, 不必进行定义。

OMSI PASCAL语言对ALFA的规定与标准PASCAL语言相同, 即规定N=10。但是, 在IBM 360/370计算机系统中使用的PASCAL 8000语言规定N为8, 请读者在使用过程中注意具体的规定。

对于字符串, 有许多灵活的应用。

首先, 允许将一个字符串定义为一个常量。例如:

CONST

C='IT IS A BOOK';

其次, 对于字符型数组变量, 不仅可以通过赋值语句进行赋值, 还可以通过读语句输入或写语句输出。例如, X为ALFA类型的变量。则下列语句都是合法的:

$X := 'XIAOLIGOOD';$

```
READLN(X);
WRITELN('X=', X);
```

最后，两个字符串可以进行关系运算。可以比较它们的大小或是否相等。例如：

1. 'ABCD'='ABCD' 为 TRUE;
2. 'ABCD'='BCDA' 为 FALSE;
3. 'ABCDEF'>'ABCDAF' 为 TRUE;

只有两个字符串的长度相同，内容相同（字符与它们的顺序都相同）时，两个串才算相等。如例 1 所示。如果有一个不相同，则为不等。如例 2 所示。

如果两个串的前 M 个字符全相同，($M \geq 0$)，第 M+1 个字符不同，则可以判断两个串的大小。如例 3 所示，'ABCD' 相同，'E'>'A'，所以第一串大于第二串。

必须指出，对字符型数组进行赋值或通过读语句输入有一些不同的特点。使用时应该注意这一点。具体不同可在下一个程序中看到。

例 7-10 综合上述字符型数组的特点，编制一个程序，并运行之。
程序如 FIG ARY04 PROGRAM 所示。

```
PROGRAM ARY04 (INPUT, OUTPUT),
CONST
  C1='PROGRAM',
  C2='SIN',
  C3='4567',
VAR
  A1, A2:ARRAY (1..7) OF CHAR,
  B1, B2:BOOLEAN,
  CLASS:ALFA,
  I :INTEGER,
BEGIN
  FOR I:=1 TO 4 DO
    BEGIN
      READLN (CLASS);
      WRITELN('CLASS', I:2, '=' , CLASS , '')
    END,
    CLASS:='COMPUTE801',
    WRITELN('CLASS A=', CLASS, ''),
    CLASS:='J801',
    WRITELN('CLASS B=', CLASS, ''),
    CLASS:=' J801',
    WRITELN('CLASS C=', CLASS, ''),
    CLASS:=' J 8 0 1 ',
    WRITELN('CLASS D=', CLASS, ''),
    A1:='ABCDEFGH',
    A2:='ABCDXYZ',
    B1:=A1<A2,
    B2:=A1=C1,
```

```

        WRITELN('B1=', B1);
        WRITELN('B2=', B2);
    END.
INPUT,
COMPUTE801
J801
    J801
J801
OUTPUT,
CLASS 1='COMPUTE801'
CLASS 2='J801'
CLASS 3='J801'
CLASS 4='J'
CLASS A='COMPUTE801'
CLASS B='J801'
CLASS C='      J801'
CLASS D='J801'
B1= TRUE
B2= FALSE
FIG ARY04 PROGRAM

```

在这个程序中，有以下四个问答比较重要，使用时必须注意：

1. 常量字符串的意义。

用一个常量代表一串字符，表示这个常量由这一串字符组成，这些字符没有其它功能。

例如：

C1与保留关键字 PROGRAM，C2与标准关键字 SIN，C3与整数 -4567 没有任何关系。

2. 字符串的运算。

字符串可以进行赋值运算，如 A1 和 A2。还可以进行关系运算，如 B1 和 B2。但是，在进行赋值运算时，必须严格地满足字符串的长度。如果用如下赋值语句：

```
A1:='ABCD';
```

则在编译过程中指出错误的性质是“类型不相容”(INCOMPATIBLE TYPE)。这是因为变量 A1 应该为 7 个字符的数组。而 'ABCD' 只有四个字符，表明实际情况与类型定义相矛盾。

3. ALFA 类型的应用

ALFA 类型是一种非常有用的标准数据类型。对 ALFA 型变量进行读、写和赋值都非常方便。初学者在开始阶段往往不太习惯用 ALFA 类型。其实，它的使用是很灵活的。

对 ALFA 型变量赋值时，一定满足 10 个字符的要求。宁可用空格补充也不能少写一个，否则同样是“类型不相容”的错误。

对 AFAL 型变量输入时，如果正好连续输入 10 个非空格字符，那么输出的结果与输入相同。如果连续输入的非空格字符少于 10 个，那么输出的结果是前几列与输入相同，后几列补充空格，满足列数总数为 10。如果开始输入的字符是空格字符，后来连续输入非空格字符(少于或等于 10 个)，那么输出的结果为清除了空格的其它非空格字符，并在其后补充空格一直满

足字符为10个为止；如果连续输入的字符中包含有空格字符，那么输出的结果为连续输入的
第一个空格字符前的非空格字符，并在其后补充空格字符到字符数为10个为止，而对空格以
后的字符不再计入，即空格字符也是ALFA类型数据的结束符。这四种情况就是程序运行后
输出的CLASS 1到CLASS 4的结果。

输出ALFA型变量时，不管这些字符来历如何，它一律按10个字符输出。

思考题：如果在执行语句READLN (CLASS) 时，输入IT IS A BOOK，那么在执行
语句 WRITELN (CLASS) 时，会输出什么？

2. 如果在执行语句READLN (CLASS) 时输入THISISABOOK，那么在执行语句
WRITELN (CLASS) 时，会输出什么？

产生上述输入、输出现象的原因在于 OMSI PASCAL-1语言的规定。

OMSI PASCAL-1 语言允许通过读语句，输入任何长度的字符数组。它把空格字符或
行结束符作为结束输入的标志。如果输入的字符个数比数组长度短，那么字符串的左侧用空
格字符来填补。反之，输入字符串太长，那么超出的字符会被截去，以满足长度的要求。

紧缩型数组，是 PASCAL 语言中规定的另一种特殊数组。通常，整数和实数型元素分
别在计算机中占一个字长和两个字长。数组的元素在计算机存储器里以连续字的方式存贮。
因此，这是一种有效的存贮方式。然而，存贮其它类型（如字符型，布尔型，子界型的变量
时，不一定是一个好方式，因为它浪费内存单元。

为了节省存贮空间，我们把数组的几个元素紧缩到一个字长中（即一个内存单元），这个
工作由编译程序去完成。在定义数组类型时，在ARRAY前面加一个PACKED就可以了。

例如：

VAR

S: ARRAY [1..1000] OF CHAR;

在 PDP-11/03 计算机系统中，一个字符占一个字节，半个字长，即 8-bit。S 这个数组
有1000个字符，它占据内存500个单元。

但是，如下的数组：

VAR

S: PACKED ARRAY [1..1000] OF CHAR;

在 CDC6000 系列机中，只要占据内存100个单元。因为它的字长为60位，每个字可以存
10个字符。每个字符只要占六位就可以了，这是由于 CDC 的字符集只有 64 个字符。但在
IBM360/370 计算机系统中，它要占据内存250个单元，这是由于IBM的字符集EBCDIC码有
72个字符，每个字符占 8 位，而系统的字长是32位。

紧缩型数组在程序中的用法与普通数组相同。紧缩型数组的运行速度比较慢。这是因为
紧缩存贮分配需要时间。因此，它是通过牺牲时间换取空间的节省。一个数组是否需要紧缩
取决于许多因素：存储器的大小，处理的速度以及数据的类型及多少等等。用户必须综合平
衡各种因素后再确定是否紧缩。

PACK 和 UNPACK 是 PASCAL 语言规定的与紧缩型数组操作有关的两个标准过
程。

在许多情况下，为了减少存取紧缩数组所需要的附加次数，人们通过一种运算，统一紧
缩或松展所有的元素。这就是标准过程PACK 和 UNPACK 的功能。

例如，我们从一个文件中读文本，并且希望把文本的代码存贮到变量字中，变量字是20

个字符的紧缩型数组。我们不是直接把字符从文件送进字，而是先将它们送进一个中间变量B，临时缓冲一下，这时并不紧缩；变量说明如下：

VAR

B:ARRAY [1..20] OF CHAR;

V:PACKED ARRAY [1..20] OF CHAR;

当变量B包含一个完整的代码时，我们调用紧缩过程PACK (B, I, V)，它将B中的所有字符紧缩到V中去。相反的是松展过程UNPACK (V, B, I)，它将V中的字符展开到B中去。

在一般情况下，如果我们有数组变量A和P，元素类型相同都为T类型，如：

VAR

A:ARRAY [M..N] OF T;

P:PACKED ARRAY [1..J] OF T;

其中M, N, I和J都是常量，一般为整数，并且满足 $M \leq N \leq J - I + N$ ，这时语句：

FOR K:=1 TO J DO P(K):=A(K-I+M);

可以简化为：

PACK (A, M, P);

它表示把数组A中的(J-I+1)个元素紧缩到紧缩型数组P中去。

同样，语句：

FOR K:=1 TO J DO A(K-I+M):=P(I);

可以简化为：

UNPACK(P, A, M);

它完成把存在紧缩型数组P中的全部元素松展到数组A中去。

必须指出，标准PASCAL语言和PASCAL8000语言中都有紧缩型数组以及标准过程PACK和UNPACK。但是OMSI PASCAL-1语言版本在处理数组问题时，虽然允许定义紧缩型数组，如：

VAR

P:PACKED ARRAY [1..50] OF CHAR;

但是它并不考虑紧缩，使用时与如下定义：

VAR

P:ARRAY [1..50] OF CHAR;

效果完全相同。在OMSI PASCAL-1语言版本中不允许使用紧缩过程PACK和松展过程UNPACK。

当数组与过程说明或函数说明结合起来使用时，有以下几点必须注意：

1. 一般数组都可以作为过程或函数的数值参数或变量参数。例如，下列类型定义和过程说明是合法的。

TYPE

M=ARRAY [1..10] OF INTEGER;

N=ARRAY [1..5] OF REAL;

PROCEDURE ABC (VAR X:M; Y, Z:N);

BEGIN

...

END;

如果改成如下形式则是非法的:

```
PROCEDURE ABC (VAR X:ARRAY [1..10] OF INTEGER,  
                Y, Z:REAL);
```

这就是说,参数的类型必须在全程的类型定义中事先定义,不允许在过程参数表中再定义类型,以保证过程的形式参数与实际参数能够兼容。

数值参数在过程调用时必须在存储器里开辟一个新的能放下整个数组的区域,而变量参数是地址传递,只用一个字保存地址就行了。因此,数组,尤其是较大的数组作为过程或函数的参数时,在可行条件下,一般用变量参数为好。这样,有利于节省存储单元和传送每个元素值的时间。

2. 一个函数的函数值不允许属于数组类型。例如,下列函数说明是错误的:

```
FUNCTION PQR (X:INTEGER):ARRAY[1..5] OF REAL;
```

即使改成如下形式,也是不允许的:

TYPE

```
FUN=ARRAY [1..5] OF REAL;
```

```
FUNCTION PQR (X:INTEGER):FUN;
```

3. 紧缩型数组不允许作为过程或函数的变量参数。

这些规定很重要,在编制程序时必须注意。否则,即使在编译时指出了语法错误,也难以找到解决问题的办法。

最后再说明一点,本节所讲到的对字符串的有关规定与操作,它主要是针对在PDP-11/0₃计算机上运行的OMSI PASCAL-1语言讲的。不同版本的PASCAL编译程序,在对字符串的处理上可能存在重大差别,这也许是最难实现“标准化”的部分。有些版本对字符串的处理作出了与此完全不同的规定,如不把空格字符作为结束字符串的标志。某些版本大大地扩充了对字符串的处理功能,这对解决某些问题是很必需的。所以,读者在自己的系统上运行某种PASCAL语言版本时,务必预先阅读有关这一部分的具体规定,以免走不必要的弯路。

§3 记录类型

一、记录类型的一般概念 几乎所有的算法语言都有数组类型,但很少有记录类型。记录类型的数据结构在PASCAL中的应用却十分广泛,它类似于PL/1语言中的STRUCTURE,有记录类型是PASCAL语言的一个重要优点。

记录类型是一种比较复杂但非常灵活的构造型数据类型。它是由固定数量的元素(又称为域)组成。元素的数量必须固定,这与数组类型相同。但是,元素的类型允许不同,这是与数组类型的一个重要区别。正是由于这一点,决定了访问它的元素的方法与数组类型完全不同。

例如,为了定义一个表示由年,月,日三项组成的日期的数据类型DATE,可以通过如下方法进行类型定义:

TYPE

DATE=RECORD

YEAR:INTEGER;

MONTH:(JAN, FEB, MAR, APR, MAY, JUN,
JUL, AUG, SEP, OCT, NOV, DEC);

DAY:1..31

END;

其中:

DATE类型由三个元素组成: YEAR, MONTH和DAY;

这三个元素分别属于三种数据类型:

YEAR 属于整数类型;

MONTH 属于枚举类型, 括号中的标记符是英文一月至十二月的缩写;

DAY属于子界类型。它表示可以1日到31日。

如果有两个变量X和Y, 它们都属于DATE类型, 则可以定义如下:

VAR

X, Y:DATE;

例如, 用X表示1949年10月1日, 用Y表示1981年7月1日。可以通过如下赋值语句来实现:

X.YEAR :=1949;

Y.YEAR :=1981;

X.MONTH :=OCT;

Y.MONTH :=JUL;

X.DAY :=1;

Y.DAY :=1;

从这些赋值语句可以看出, 表示一个记录元素与表示一个数组元素完全不同。数组元素是通过变量名与相应下标来表示的, 而记录元素是通过变量名与域名来表示。应把变量名与域名用一个圆点连起来。

当然, 由于月和日同样可以用整数表示, 把DATE定义为一个数组类型也是允许的。例如,

TYPE

DATE=ARRAY [YEAR, MONTH, DAY] OF INTEGER;

VAR

X, Y : DATE;

表示1949年10月1日和1981年7月1日可以用如下赋值语句:

X [YEAR] :=1949;

Y [YEAR] :=1981;

X [MONTH] :=10;

Y [MONTH] :=7;

X [DAY] :=1;

Y [DAY] :=1;

在上面的例题中, 只是由于记录类型DATE中的MONTH和DAY可以用整数表示, 可以扩展成整数, 因此, 数组类型DATE能够代替记录类型DATE, 不影响程序设计。

如果要记录某个小组四位同学各方面的情况: 学号、姓名、性别、年龄和学习成绩, 显而易见, 姓名、性别和年龄是无法用同一种数据类型表示的, 因此, 不能定为数组类型, 必须定义为记录类型。例如:

TYPE


```

STUDENT=RECORD
    I :INTEGER;
    NAME:(ZHAO, QIAN, SUN, LI);
    SEX : (MALE, FEMALE);
    AGE :18..25;
    SCORE:REAL
END;

```

其中:

STUDENT 类型由五个元素组成: I, NAME, SEX, AGE, SCORE;
这五个元素分别属于五种数据类型:

I 表示学号, 属于整数类型;

NAME 表示姓名, 这里仅表示姓。属于枚举类型。包含赵、钱、孙、李四位同学;

SEX 表示性别, 属于枚举类型。包括男生和女生两种;

AGE 表示年龄, 属于子界类型。允许从18岁到25岁;

SCORE 表示成绩, 属于实数类型。

例7-11 编制一个程序记下男生小赵(学号1001号, 22岁, 成绩85.5分)和女生小李(学号1004号, 18岁, , 成绩83分)的情况。

程序的FIG RCD01 PROGRAM 所示。

```

PROGRAM RCD01(INPUT,OUTPUT);
TYPE
STUDENT=RECORD
    I :INTEGER;
    NAME:(ZHAO,QIAN,SUN,LI);
    SEX:(MALE,FEMALE);
    AGE:18..25;
    SCORE:REAL
END;

VAR
    S1,S2:STUDENT;
BEGIN
    S1.I:=1001;          S2.I:=1004;
    S1.NAME:=ZHAO;       S2.NAME:=LI;
    S1.SEX:=MALE;        S2.SEX:=FEMALE;
    S1.AGE:=22;          S2.AGE:=18;
    S1.SCORE:=85.5;      S2.SCORRE:=83
END.

FIG RCD01 PROGRAM

```

在解决各种实际问题编写相应程序时, 记录是用得很普遍, 用起来既方便又清晰的一种构造类型数据。

在用PASCAL语言写PASCAL的编译程序, 以及学习一个机器的操作系统时, 记录是一个非常重要的数据类型。

二、记录类型说明和域名 在PASCAL语言中,对记录类型进行说明时,通常将RECORD和END之中间部分叫做域表 (FIELD LIST);把I, NAME, SEX等称为域名 (FIELD NAME);域名后面给出的是域的类型。

记录类型说明的一般格式如图7.3-1所示。

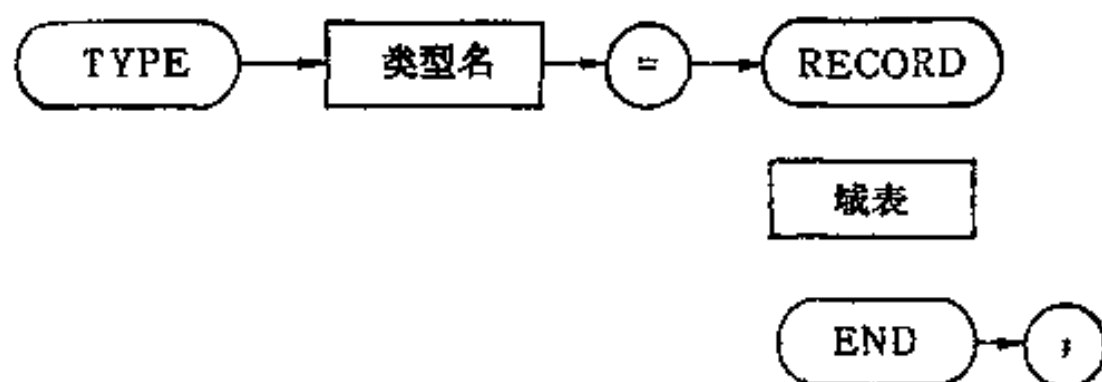


图7.3-1 记录类型说明的一般格式

其中域表的一般格式如图7.3-2:

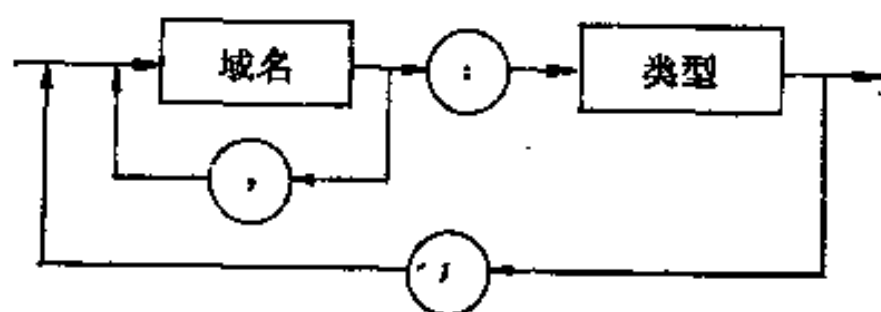


图7.3-2 域表的格式

在PASCAL语言中,保留关键字END用途很广,综合起来有如下四条——
程序执行部分的BEGIN.....END.

复合语句中的BEGIN.....END;

情况语句中的CASE.....END;

记录类型定义中的RECORD.....END;

它们表明,END一般不会单独出现,总是与其它关键字成对出现的。在不同的地方,END的意义各不相同。用户在使用时,尤其在编写长程序时,最容易犯的错误是遗漏掉END。

记录类型变量 (简称记录) 说明的格式如图7.3-3所示。

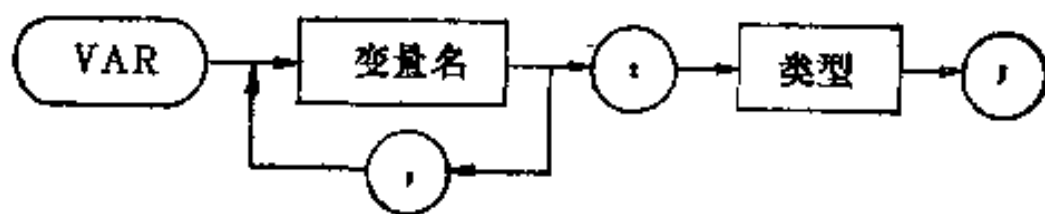


图7.3-3 记录类型变量说明的格式

表示一个记录元素的格式如图7.3-4

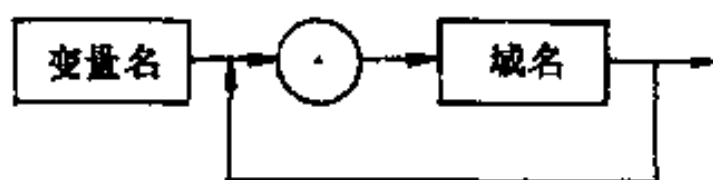


图7.3-4 记录元素的格式

如果在类型说明时，在RECORD之前加上关键字PACKED，那么表示是紧缩型记录类型。其基本思想与紧缩型数组类型相同，同样是通过牺牲存取时间来节省存储单元。在一般情况下，紧缩型记录的元素不能作为过程或函数的参数。

将记录类型说明和变量说明合并起来是允许的。但是，由于记录类型比较复杂，单独说明更加清楚。

域名和域的类型的关系与变量名及其对应类型之间的关系相同。允许几个域名同属于一种类型，但不允许一个域名从属于几种类型。

一个域名可以是任何类型，甚至是一个记录类型。例如，记录学生情况的记录类型STUDENT中，成绩SCORE本身可能是一个记录：记录政治，数学，物理，英语和平均分。

为此，我们重新定义记录类型STUDENT如下：

```
TYPE
    STUDENT=RECORD
        I      : INTEGER;
        NAME: (ZHAO, QIAN, SUN, LI);
        SEX : (MALE, FEMALE);
        AGE : 18..25;
        SCORE: RECORD
            POLITICS, MATHEMATICS,
            PHYSICS, ENGLISH: INTEGER;
            AVERAGE : REAL
        END
    END;
VAR
    S STUDENT;
```

在这种情况下，表示某个同学的某门功课的成绩时，必须用两个域名：一个是大域名，SCORE；一个是小域名，如ENGLISH。

比如，小赵的英语成绩为85分。必须用下列赋值语句：

```
S.SCORE.ENGLISH:=85;
```

例7-12 编制一个程序，表示小赵的各方面的情况。其中政治课86分，数学课85分，物理课86分，英语课85分。平均成绩85.5分。

程序如FIG RCD02 PROGRAM所示：

```
PROGRAM RCD02(INPUT,OUTPUT),
TYPE
    STUDENT=
    RECORD
        I      : INTEGER;
        NAME : (ZHAO, QIAN, SUN, LI);
        SEX   : (MALE, FEMALE);
        AGE   : 18..25
        SCORE:
        RECORD
```

```

        POLITICS, MATHEMATICS,
        PHYSICS, ENGLISH: INTEGER,
        AVERAGE: REAL
    END
END,
VAR
    S: STUDENT;
BEGIN
    S.I := 1001;
    S.NAME := ZHAO;
    S.SEX := MALE;
    S.AGE := 22;
    S.SCORE.POLITICS := 86;
    S.SCORE.MATHEMATICS := 85;
    S.SCORE.PHYSICS := 86;
    S.SCORE.ENGLISH := 85;
    S.SCORE.AVERAGE := 85.5
END.
FIG RCD02 PROGRAM

```

如果记录类型中域名的类型是数组类型，使用起来也是非常方便的。例如，可以将学生四门功课的成绩用一个包含四个整数类型元素的数组来表示。平均成绩仍然用实数表示。重新定义类型STUDENT如下，

```

TYPE
    STUDENT = RECORD
        I      : INTEGER;
        NAME: (ZHAO, QIAN, SUN, LI) ;
        SEX   : (MALE, FEMALE);
        AGE   : 18 .. 25;
        SCORE: RECORD
            A : ARRAY (1 .. 4) OF INTEGER;
            AVERAGE : REAL
        END
    END;
VAR
    S : STUDENT;

```

例7-13 编制一个程序表示小赵各方面的情况。其中四门功课的成绩用数组表示，平均成绩用实数表示。

程序如FIG RCD03 PROGRAM所示。

为了程序清晰整齐，将ARRAY (1 .. 4) OF INTEGER定义为类型ABC。

```

PROGRAM RCD03 (INPUT, OUTPUT) ,
TYPE
    ABC = ARRAY (1 .. 4) OF INTEGER;

```

```

STUDENT=
RECORD
    I      : INTEGER;
    NAME   : (ZHAO, QIAN, SUN, LI);
    SEX    : (MALE, FEMALE);
    AGE    : 18..25;
    SCORE  : RECORD
                A      : ABC;
                AVERAGE: REAL
    END
END;

VAR
    S: STUDENT;
BEGIN
    S.I := 1001;
    S.NAME := ZHAO;
    S.SEX := MALE;
    S.AGE := 22;
    S.SCORE.A (1) := 88;
    S.SCORE.A (2) := 85;
    S.SCORE.A (3) := 86;
    S.SCORE.A (4) := 85;
    S.SCORE.AVERAGE := 85.6
END.

```

FIG RCD03 PROGRAM

三. 开域语句 “开域”, 就是“打开疆域”的意思。它有效地打开了指定的记录变量的域名的辖域, 使得域名可以象变量名那样灵活地使用。

例如, 让记录变量X表示1949年10月1日, 可以用下述语句。

```

WITH X DO
BEGIN
    YEAR := 1949;
    MONTH := OCT;
    DAY := 1
END;

```

这表示, 打开X的“疆域”, YEAR、MONTH和DAY都获得了“解放”, 书写和阅读都比较方便了。开域语句的格式如图7-3-5。

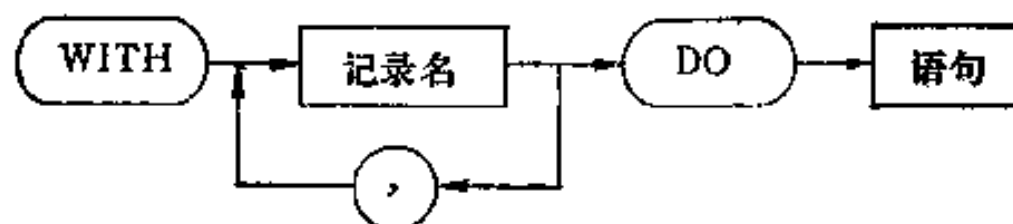


图7.3—5 开域语句的格式

开域语句不仅可以打开记录变量，还可以打开大的域名。比如，为了表示男生小赵的详细情况，不仅可以打开 S1，还可以打开 SCORE。

例7-14 编制一个程序表示小赵和小李的情况。要求都用开域语句。小赵的成绩用赋值语句实现，小李的成绩用读语句实现。

程序如FIG RCD04 PROGRAM所示。

```
PROGRAM RCD04(INPUT, OUTPUT),
TYPE
  ABC=ARRAY (1..4) OF INTEGER,
  STUDENT=RECORD
      I      : INTEGER,
      NAME   : (ZHAO, QIAN, SUN, LI),
      SEX    : (MALE, FEMALE),
      AGE    : 18..25,
      SCORE  : RECORD
          A      : ABC,
          AVERAGE: REAL
      END
  END,
VAR
  S1, S2: STUDENT,
  K, SUM: INTEGER,
BEGIN
  WITH S1 DO
    BEGIN
      I      := 1001,
      NAME   := ZHAO,
      SEX    := MALE,
      AGE    := 22,
      WITH SCORE DO
        BEGIN
          A (1) := 86,
          A (2) := 85,
          A (3) := 86,
          A (4) := 85,
          AVERAGE := 85.5
        END
      END,
    WITH S2, SCORE DO
      BEGIN
        I      := 1004,
        NAME   := LI,
        SEX    := FEMALE,
        AGE    := 18,
```

```

        FOR K:=1 TO 4 DO
            BEGIN
                READ (A [K] );
                SUM:=SUM+A [K] ;
            END;
        AVERAGE:=SUM/4
    END
END.
FIG RCD04 PROGRAM

```

在这个程序中，由于学生的成绩是整数，可以直接输入和输出。因此，学生的成绩既可以用赋值语句一个个确定。如小赵的成绩。也可以用读语句逐个输入，然后求和，计算平均成绩。如小李的成绩。

在这个程序中使用开域语句时用了两种形式。对于小赵，用的是如下开域语句，

```

WITH S1 DO
    BEGIN
        语句 1;
        WITH SCORE DO 语句 2
    END;

```

对于小李，用的是如下开域语句，

```

WITH S2, SCORE DO
    BEGIN
        语句 3
        语句 4
    END;

```

这两种形式具有完全等效的功能。因为S1和SCORE或S2和SCORE之间的关系不是并列的关系，而是嵌套的关系。

语句：

```
WITH R1, R2, R3...Rn DO 语句
```

完全等价于如下语句：

```

WITH R1 DO
    WITH R2 DO
        WITH R3 DO
            .
            .
            .
            WITH Rn DO 语句

```

一般地说，无限次嵌套是不允许的。在 OMSI PASCAL-1 语言中，开域语句的嵌套最多允许四层。否则无法编译。

开域语句是一种构造型语句。它的成分语句通常叫限定语句。如果成分语句超过一个时，必须用复合语句。限定语句不允许改变已开域记录。就是说，在开域语句

```
WITH R DO 语句 S
```

执行时，语句 S 不能影响 R。

若开域语句：

```
WITH A [1] DO
```

```
BEGIN
```

```
语句 1
```

```
I := I + 1
```

```
END;
```

这是不允许的。因为 $I := I + 1$ 改变了 $A[1]$ ，所以不能正常运行。

当然，非限定性语句可以改变记录。例如语句

```
FOR I:=1 TO 10 DO
```

```
WITH A [I] DO 语句
```

这是合法的。因为循环语句不属于限定语句。

四、记录的变体部分 前面的例题中，由于每一个学生，都有学号，姓名，性别，年龄及各门功课的成绩和平均成绩。

在实际生活中还会遇到这样的问题。例如对某些教师希望反映他们的学位情况，即是否获得博士学位，如果是，什么时间什么地点得到这个学位的？如果是硕士学位，就搞清何时获得的就行了。每个教师的学位情况可以很不一样，这里当然有一个选择问题。在 PASCAL 语言中，解决这类问题要用到记录的变体部分。

下面是教师情况的类型定义。

```
TYPE
```

```
DEGREE = (D, M, Q);
```

```
TEACHER = RECORD
```

```
  I: INTEGER;
```

```
  AGE: 20..70;
```

```
  CASE STATUS: DEGREE OF
```

```
    D: (YEAR1: INTEGER,
```

```
        STATE: ALFA);
```

```
    M: (YEAR2: INTEGER);
```

```
    Q: (YEAR3: INTEGER);
```

```
END;
```

```
VAR
```

```
  S: TEACHER;
```

在类型 TEACHER 的定义中：

CASE 前面的部分是记录类型的固定部分，它与前面例 7-3-4 中定义的 STUDENT 类型没有本质的区别。

CASE 后面的部分是记录类型的变体部分，它使用户有选择域名的自由。

变体部分的 OF 之前有两项：标志域名 (STATUS) 和标志域类型 (DEGREE)。其中标志域类型必须为枚举类型。

变体部分的 OF 之后表示记录类型由三个变体组成。

每一个变体由一个或几个情况标号和域表组成。域表必须在一对圆括号之中。变体部分结束时的END与记录类型定义结束的END是公用的。如果将标志域名及其类型放在CASE前面也是允许的。下述方法也是正确的。

```
TYPE
  TEACHER=RECORD
      I      :INTEGER
      AGE    :20..70;
      STATUS:DEGREE;
      CASE DEGREE OF
          .
          .
          .
      END ;
```

如果对于某个标号，如Q，虽然它的域表是空的，则在圆括号内没有任何内容。但是，在 OMSI PASCAL—1语言中，域表的内容不允许是空的。否则，编译时指出错误的性质是“域表有错” (BAD FIELD LIST);因此，尽管在程序中并不需要这个域表，仍需要确定一个域表。例如，定义：Q:(YEAR3:INTEGER);

在同一个记录类型中，所有的域名必须是不同的。即使它们在不同的变体中，甚至分别在记录类型的固定部分和变体部分中，否则编译时指出错误的性质是“域名重复” (DUPLICATE FIELD NAME)。当然，在不同的记录类型中，允许使用同样的域名。

在一个记录类型中，只允许有一个变体部分。变体部分必须在固定部分之后。但是，变体部分本身可以包含变体。也就是说，变体允许嵌套。

图7.3—6是记录类型定义的更完整的格式。

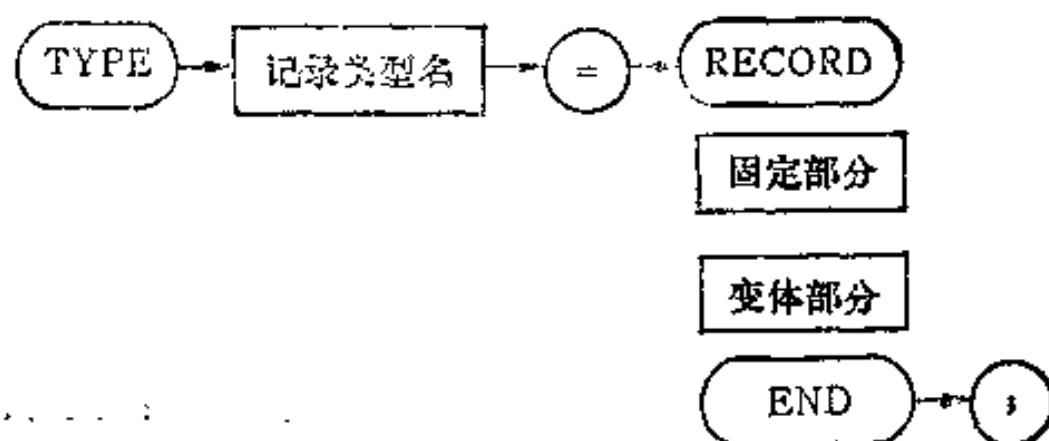


图7.3—6 记录类型完整的格式

其中变体部分的格式如图7.3—7。

为了帮助读者掌握关于记录类型变体部分的内容，下面举一个简单的程序作为例子，从中弄清有关的规则和使用方法。

例7-15 编制一个程序，输入某科研小组成员情况的某些简单数据，主要是年龄和学位，进行处理并输出。

为简单起见，成员只有4个，且输入一个处理一个。输入数据，首先是年龄，按着是代表学位的字符，如果是博士学位，则要再输入获得此学位的时间，还要输入代表获得学位地点的字符，表示博士学位是在国外还是在国内获得的。如果是硕士学位，则再输入获得时间就行了。如果未有学位，则不必再输入任何内容。程序如FIG RCD05 PROGRAM所示。

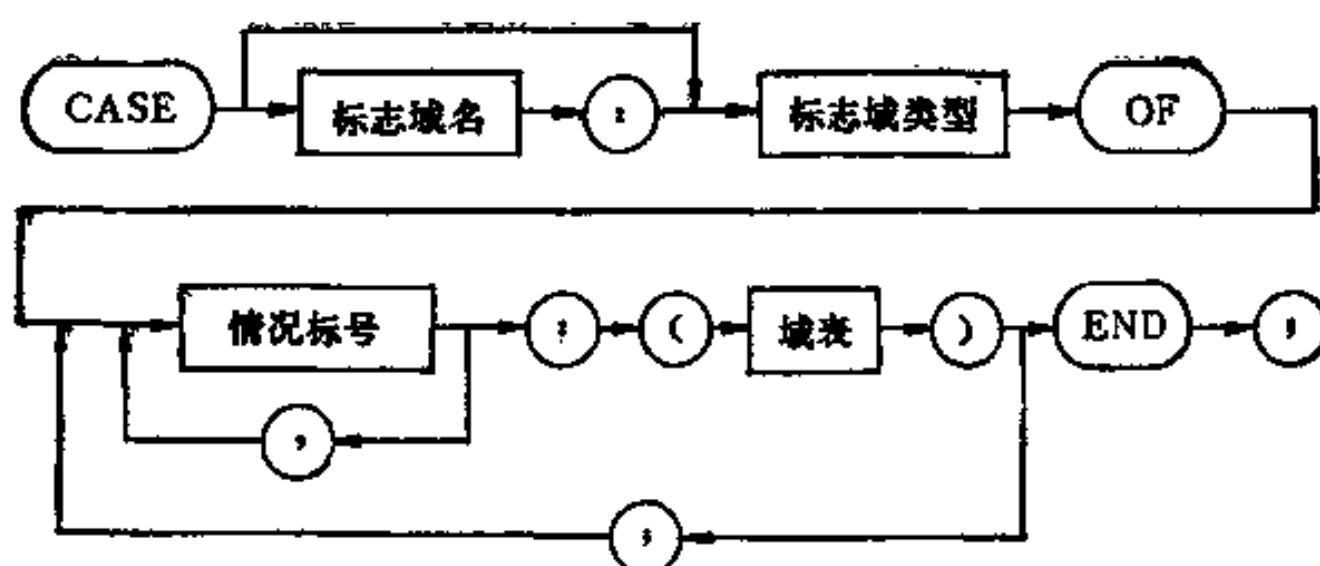


图7.3—7 记录类型变体部分的格式

```

PROGRAM RCD05(INPUT, OUTPUT);
LABEL
  10;
TYPE
  DEGREE=(D, M, Q),
  TEACHER=RECORD
      I      :INTEGER,
      AGE    :20..70;
      CASE STATUS: DEGREE OF
          D      :(YEAR1:INTEGER,
                  STATE:ALFA),
          M      :(YEAR2:INTEGER),
          Q      :(YEAR3:INTEGER),
      END;
VAR
  S : TEACHER,
  CH : CHAR,
  K:INTEGER,
BEGIN
  WRITELN('INPUT AGE, DEGREE, DATE, LOCATION,')
  WRITELN,
  WRITELN(' NO.    AGE      ( STATUS OF DEGREE ) ', )
  WRITELN('          DEGREE DATE  LOCATION'),
  FOR K:=1001 TO 1004 DO
  WITH S DO
  BEGIN
    READ(AGE),
    I:=K,
    READ(CH),
    CASE CH OF
      'D': STATUS:=D,
      'M': STATUS:=M,
      'Q': STATUS:=Q,
    ELSE

```

```

      BEGIN
        WRITELN('STATUS OF DEGREE ERROR'),
        GOTO 10
      END
    END,
CASE STATUS OF
D:
  BEGIN
    READLN(YEAR1,CH),
    IF CH='F'
      THEN STATE:='FROMABROAD'
      ELSE STATE:='AT-HOME',
    WRITE(I:5,AGE:6),
    WRITELN(' PH.D.',YEAR1:7,' ',STATE),
  END,
M:
  BEGIN
    READLN(YEAR2),
    WRITE(I:5,AGE:6),
    WRITELN(' M.S. ',YEAR2:7)
  END,
Q:
  BEGIN
    WRITE(I:5,AGE:6),
    WRITELN(' OTHER')
  END
END
END,
10:
  END.
INPUT:
56D1951F
45D1981S
25M1982
22Q
OUTPUT:
NO.   AGE   ( STATUS OF   DEGREE )
          DEGREE DATE   LOCATION
1001   56   PH.D.    1951   FROMABROAD
1002   45   PH.D.    1981   AT-HOME
1003   25   M.S.     1982
1004   22   OTHER

```

FIG RCD05 PROGRAM

在这个程序中，为简化起见，对人员的其它情况，如姓名、性别、工资或成绩等不予考

虑,重点抓住学位的情况,保留编号及年龄。

记录类型变体部分的应用,在程序的执行部分看得很清楚。在本程序中,变体情况语句来实现,而且用了两个情况语句来完成。

第一个情况语句解决学位的选择问题,由于DEGREE是枚举类型,不能直接通过语句输入。因此,得先输入特征字符,然后转换为枚举类型数据。在这里,D代表博士学位(PH.D.),M代表硕士学位(M.S.),Q代表没有学位的成员(OTHER)。如果输入的字符不是D,M和Q,则表示输入的学位情况有错。

第二个情况语句,进一步解决何时何地获得学位的问题。由于STATE是ALFA类型,这里为简化输入地点起见,只需输入一个特征字符即可,输入字符F,表示博士学位是在国外获得的,输入其它字符,则表示博士学位是在国内授予的。

由于学位情况不同,输出的项目也不相同。请注意的是,尽管科研小组四个成员情况在变体部分有所不同,但它们都是相同类型的数据。

思考题:

1. 在本程序中是如何解决终端输入字符时自动加入的空格字符问题的?
 2. 为什么程序中用READ (AGE); READ (CH); READLN (YEAR, CH),而不用下列语句: READLN (AGE); READLN(CH); READLN(YEAR, CH);与数据输入时的格式有无联系?
 3. 为什么程序执行时,输入的数据是一个接一个,中间没有隔开?
- 如果将56D1951F改成56 D 1951 F,行不行?为什么?

大家知道,在一个程序中,一个变量名不能属于两种数据类型,否则,在编译时会出现错误。例如:

```
VAR
    M : 1..10;
    M : ARRAY ('A'..'F') OF REAL;
```

在一个记录类型里,元素的名称(域名)必须是唯一的,不允许一个域名属于两种数据类型。

但是,对两(或两个以上)不同的记录类型,这个记录的域名也可以作为其它记录类型的域名,还可以作为变量的名称。这是记录应用中的灵活性。

例7-16 综合上述概念,编制一个程序如 FIG RCD06 PROGRAM 所示

```
PROGRAM RCD06(INPUT,OUTPUT);
TYPE
    P=RECORD
        M:ARRAY ('A'..'F') OF INTEGER;
        N:BOOLEAN;
    END;
    Q=RECORD
        P:(A,B,C,D,E,F);
        N:50..100;
    END;
VAR
    SUM:INTEGER;
```

```

CH: 'A' .. 'F',
M: P,
N: Q,
BEGIN
  READLN;
  READLN(CH);
  READLN(M.M (CH) );
  READLN(N.N);
  M.N:=5>3;
  N.P:=C;
  SUM:=M.M (CH) +N.N;
  WRITELN('M.N=',M.N);
  WRITELN('SUM=',SUM:6)
END.
INPUT:
B
10
60
OUTPUT:
M.N= TRUE
SUM= 70
INPUT:
W
OUTPUT:
ARRAY BOUNDS ERROR
FIG RCD06 PROGRAM

```

在这个简单程序的类型定义和变量说明中，多次使用M、N和P 而且各自意义不同。其中：

M既表示P类型的一个记录类型变量，又代表P类型的一个域名，属于数组类型。域名与变量名的功能完全不同，要注意区别。

N表示Q类型的一个记录类型变量。它又表示P类型的一个域名，属于布尔类型。它还表示Q类型的一个域名，属于子界类型。虽然字母N是“一身三任”，但是并不矛盾。

P、Q分别表示一个记录类型。

P还表示Q类型的一个域名，属于枚举类型。这也是允许的。

总而言之，在记录的不同范畴中，用同一个标识符可以表示不同的功能。当然，在程序设计时，不用相同的标识符更便于阅读。在复杂的情况下，可以避免不必要的混淆与麻烦。

由于类型定义的合法性，因此，M.M (CH)，N.N和N.P都是允许的

从这个程序的执行部分可以看出，对于记录类型变量，可以进行赋值操作。例如：

M.N:=5>3;

N.P:=C;

记录类型变量可以直接通过读语句输入，也可以通过写语句输出。例如：

READLN (M.M(CH));

```
READLN (N.N);
WRITELN ('M.N=',M.N);
```

当然，如果记录类型的域名属于枚举类型，那么这种记录变量是无法直接输入和输出的。否则，编译时显示错误为“读语句有错” (“BAD READ STATEMENT”)或写语句有错” (“BAD WRITE STATEMENT”)。例如，

```
READLN (N.P);
WRITELN (N.P);
```

此外，域名属于布尔类型时，只能直接输出，不能直接输入。例如，

```
READLN (M.N); 是不行的。
WRITELN (M.N); 是允许的。
```

由于输入的字符W不是数组M.M的下标，因此，运行时指出“数组下标越界错误” (ARRAY BOUNDS ERROR)。

在PASCAL语言中，如果两个变量属于同一种记录类型，允许通过一个赋值语句把一个记录型变量的各个域名的数值对应地赋给另一个记录型变量的域名。例如，

```
TYPE
    DATE=RECORD
        YEAR : INTEGER;
        MONTH : (JAN, FEB, ...DEC);
        DAY : 1..31
    END;
VAR
    X, Y : DATE;
```

那么语句

```
Y:=X;
```

等于下列三个语句的功能：

```
Y.YEAR := X.YEAR;
Y.MONTH := X.MONTH;
Y.DAY := X.DAY;
```

前面讨论的都是记录类型的元素类型问题。反过来，记录类型也可以作为数组类型的元素类型。例如，表示一个小组4个学生情况时，可以定义类型GROUP，

```
TYPE
    STUDENT=RECORD
        域表
    END;
    GROUP=ARRAY [ 1..4 ] OF STUDENT;
VAR
    G : GROUP;
```

在这个类型定义中，GROUP是由四个元素组成的数组类型。元素的类型是STUDENT，如例7-13中它由五个元素组成的记录类型。其中有的元素（域名）的类型又是记录类型。如SCORE。当然，SCORE的域名A又可以是数组类型。这就是，记录类型与数组类型交

替使用会给用户带来很大的方便。

如果希望表示该组第一个学生小赵的物理课成绩86分, 可以用如下赋值语句:

```
G [ 1 ] .SCORE.PHYSICS:=86;
```

或者用数组表示如下:

```
G [ 1 ] .SCORE.A [ 3 ] :=86;
```

例7-17 通过读语句输入该小组四位同学各门功课的成绩。然后计算平均成绩。最后, 输出一份学生成绩表。

分析:

为了突出主要矛盾——学生成绩, 放弃了难以输入输出的姓名和性别, 保留了学号及年龄。

输入及输出时都用双重循环语句。外层循环解决四位同学的循环问题, 内层循环解决四门功课的循环问题。

为了清楚地表明一个数据的从属关系, 输入数据时不用开域语句表示。但是, 输出数据时用开域语句。

程序如FIG RCD07 PROGRAM 所示。

```
PROGRAM RCD07(INPUT,OUTPUT),
TYPE
  ABC=ARRAY (1..4) OF INTEGER,
  STUDENT=RECORD
      I      :INTEGER,
      NAME  :(ZHAO,QIAN,SUN,LI),
      SEX   :(MALE,FEMALE),
      AGE   :18..25,
      SCORE :RECORD
          A      :ABC,
          AVERAGE:REAL
      END
  END,
  GROUP=ARRAY (1..4) OF STUDENT,
VAR
  G      :GROUP,
  M,J,SUM:INTEGER,
BEGIN
  FOR M:=1 TO 4 DO
    BEGIN
      SUM=0,
      READ(G [M] AGE),
      FOR J:=1 TO 4 DO
        BEGIN
          READ(G [M] .SCORE.A [J] ),
          SUM:=SUM+G [M] .SCORE.A [J]
        END,

```

```

        G (M) .SCORE.AVERAGE:=SUM/4
    END;
    WRITELN(' NO. AGE FOL1 MATH PHYS ENGL AVER'),
    FOR M:=1 TO 4 DO
    WITH G (M) DO
    BEGIN
        I:=1000+M;
        WRITE(I:5,AGE:5);
        FOR J:=1 TO 4 DO WRITE(SCORE A (J) :5);
        WRITELN(SCORE.AVERAGE:6:1)
    END;
    END.
INPUT:

```

```

22    85    85    86    85
21    84    86    86    84
20    87    82    82    87
18    85    83    83    85

```

OUTPUT:

NO	AGE	FOL1	MATH	PHYS	ENGL	AVER
1001	22	86	85	86	85	85.5
1002	21	84	86	86	84	85.0
1003	20	87	82	82	87	84.5
1004	18	85	83	83	85	84.0

FIG RCD07 PROGRAM

在这个程序中，变量M作为四位同学循环的控制变量。变量J作为四门功课循环的控制变量。主要变量G是GROUP类型的变量。GROUP是一个复杂的数据类型。

程序的执行部分由两大部分组成：

1. 输入部分

输入部分由一个循环语句组成。它逐个输入每位同学的年龄以及各门功课的成绩。然后求出该同学的总分及平均分。

2. 输出部分

输出部分由一个写语句和一个循环语句组成。写语句输出学生情况表的表头。循环语句计算每位同学的学号，然后逐个输出每位同学的学号、年龄、四门功课的成绩及平均分数，建立起一个完整的表。

请读者注意，记录类型数据的上述应用只是原理性的，实用价值不大。因为统计学生成绩及用作管理其它应用不可能临时输入，临时作统计。它必须输入较多的统计数字和有关材料，作为一个文件，然后进行处理。所以记录的实际应用，必须是记录和文件结合在一起，详细情况请见后面介绍。

§4. 应用举例

构造型数据类型的应用非常广泛。由于PASCAL语言不仅具有一般算法语言都具备的

数组类型，而且有集合类型、记录类型和文件类型。它们各有特点，综合应用它们，会给程序设计带来极大的方便。下面就举几个例题来说明它们的应用。

例7-12A 编制程序实现下述功能：将2到65以内的全部素数找出来，然后将求得的每五个素数排成一行，形成素数表。

分析：

什么叫素数？

素数又叫质数。它是只能被1和自己本身整除的正整数。例如，2和7都是素数，但是，8不是素数。因为它不仅能被1和8整除，还能被2和4整除。

怎样找素数？

如果I是一个正整数，则将它作为被除数，先后选择2, 3, …… I-1作为除数。如果整除后的余数都不为零，则I就是一个素数。否则，I不是素数。这个过程通常用取模运算来实现。

如何建立素数表？

逐个分析2到65中有哪些素数。如果是素数，则输出素数时进行计数。每五个素数为一行，最终形成素数表。

程序如FIG SAR01A PROGRAM 所示。

```
PROGRAM SAR01A(OUTPUT),
CONST
  N=65;
VAR
  I,J,K:INTEGER;
BEGIN
  Writeln('ALL PRIMES IN 2..65 ARE:');
  Write('2':4);
  K:=1;
  FOR I:=3 TO N DO
    FOR J:=2 TO I-1 DO      (* (I-1).....(I DIV 2) *)
      IF I MOD J=0
        THEN EXIT
        ELSE
          IF J=I-1          (* (I-1).....(I DIV 2) *)
            THEN
              BEGIN
                K:=K+1;
                Write(I:4);
                IF K MOD 5=0
                  THEN Writeln
                END
              END
            END
  END.
OUTPUT:
ALL PRIMES IN 2..65 ARE:
  2   3   5   7  11
```

```

13  17  19  23  29
31  37  41  43  47
53  59  61

```

FIG SAR01A PROGRAM

这个程序本身是正确的，但程序运行时间较长。如果把素数寻找的范围从 2..65 扩大到 2..10000，那么运行时间是 0.4869 小时。

如果把“FOR J:=2 TO I-1”中的 I-1 改为 I DIV 2（因为任何数的最大公约数不会超过这个数的 $\frac{1}{2}$ 。当然下面 IF J=I-1 也要改为 J=I DIV 2）那么运行时间为 0.2579 小时，缩短了近一半。

如果用集合的办法，见程序 SAR08 PROGRAM，运行时间不过 26.92 秒，是 0.4869 小时的六十七分之一。

这个程序没有应用集合的概念。如果改变常量 N，或者将 N 定义为变量，通过读语句输入 N，那么，它就可以求出 2 到 N 以内的素数表。

例 7-18 B 请编程序用集合的方法来实现例 7-18 A 的功能。

分析：

厄拉多塞 (ERATOSTHENES) 的筛选法是运用集合的一种好方法。

首先，定义一个集合类型。如 ABC。它包含 64 个元素。如 2 到 65。（如果超过 64 个元素，则通过设置元素为集合类型的数组或其它办法来解决）。

其次，定义两个集合变量，如筛 (SIEVE) 集合 S 和素数 (PRIME) 集合 P。它们是 ABC 类型的变量。

最后，找出全部素数，建立完整的素数表，其步骤如下：

1. 将 2 到 65 逐个放入筛中，建立筛集合 S。
2. 选定筛中最小的素数。如 2。
3. 将选定的素数放入素数集合 P 中。
4. 检查筛集合 S，从中消去选定素数和它的所有倍数。如选定 2，则从筛集合 S 中去掉全部偶数元素，如选定 3，则从筛集合中去掉全部 3 的倍数的那些数等等。
5. 重复步骤 2，3，4。直到筛集合 S 变成空集合，素数集合 P 完全建立。
6. 间接输出集合 P 中的元素，并且每五个素数为一行，形成素数表。

程序见 FIG SAR 01B PROGRAM 所示。

```

PROGRAM SAR 01B(OUTPUT);
CONST
  N=65;
TYPE
  ABC=SET OF 2..N;
VAR
  S,R:ABC;
  M,I,J,K:INTEGER;
BEGIN
  (* PART 1.....CREATE SIEVE *)
  S:= {2..65};

```

```

(• PART 2.....FIND PRIME AND DELETE SIEVE •)
P:= ( ) ,      J:=2;
REPEAT
  (• PART 2A.....FIND PRIME •)
  IF J<=N
  THEN
    BEGIN
      M:=J;
      P:=P+ [J]
    END;
  (• PART 2B...DELETE SIEVE •)
  WHILE M<=N DO
    BEGIN
      S:=S- [M] ,
      M:=J+M
    END;
    WHILE (J<=N) AND NOT (J IN S) DO J:=SUCC(J),
  UNTIL S= [ ] ,
  (• PART 3.....OUTPUT PRIMES •)
  K:=0;
  WRITELN('ALL PRIMES IN 2..65 ARE:'),
  FOR I:=2 TO N DO
  IF      I IN P
  THEN
    BEGIN
      WRITE(I:4);
      K:=K+1;
      IF   K MOD 5=0
      THEN WRITELN
    END
  END.

```

OUTPUT:

ALL PRIMES IN 2..65 ARE:

2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61		

NOTE:

IF N=650

THEN WHILE RUNNIG THE PROGRAM

DISPLAY 'BAD SET EXPRESSION'

FIG SAR01B PROGRAM

例7-19 输入一个个字符，形成一个标识符。标识符要求以英文字母开头，后跟字母或

数字符。也可以用单独一个英文字母组成一个标识符。假定标识符的长度不超过 8 个字符。但在遇到回车或空白后表示标识符输入完毕。编制程序实现如上功能。

分析。

为了提高运行速度，将字母和数字用集合来表示。标识符用八个字符组成的数组表示。因此，这个程序是集合与数组类型的综合应用。

程序如 FIG SAR02 PROGRAM 所示。

```
PROGRAM SAR02(INPUT, OUTPUT),
LABEL
  10,
VAR
  I:1..8,
  CH:CHAR,
  A:ARRAY (1..8) OF CHAR,
  LETTER:SET OF 'A'..'Z',
  DIGIT:SET OF '0'..'9',
BEGIN
  LETTER := ('A'..'Z'),
  DIGIT := ('0'..'9'),
  REPEAT
    READ(CH);
  UNTIL CH<>' ',
  I:=1,
  IF CH IN LETTER
  THEN A(I) := CH
  ELSE
    BEGIN
      WRITELN('IDENTIFIER ERROR1'),
      GOTO 10
    END,
  REPEAT
    I:=I+1,
    READ(CH);
  IF (CH IN LETTER) OR (CH IN DIGIT)
  THEN A(I) := CH
  ELSE
    IF CH=' '
    THEN EXIT
    ELSE
      BEGIN
        WRITELN('IDENTIFIER ERROR1'),
        GOTO 10
      END,
  UNTIL I=8,
```

```

        WRITELN('IDENTIFIER:', '', A, '');
10:
    END.
INPUT:                OUTPUT:
ABCDEF12             IDENTIFIER : 'ABCDEF12'
  ABCDEF1234          IDENTIFIER : 'ABCDEF12'
    ABCDEF            IDENTIFIER : 'ABCDEF '
  ABCD EF1234         IDENTIFIER : 'ABCD '
ABCDEF? XYZW          IDENTIFIER ERROR;
123EF?XYZW            IDENTIFIER ERROR;
#ABCDEF               IDENTIFIER ERROR;

```

FIG SAR02 PROGRAM

程序执行部分可以分为四部分:

1. 初始准备部分

这一部分建立字母和数字的集合, 并消去输入的全部空格字符。

2. 输入第一个字母

如果输入的第一个非空格字符是字母, 则将它作为标识符的第一个字符。否则, 作为“标识符错误”处理。

3. 输入完整的标识符

继续输入字符, 如果这些字符是字母或数字, 则将它们作为标识符的内容。当然, 最多只能接受八个字符。如果输入的是空格字符, 则表示标识符输入结束, 退出循环。否则, 作为“标识符错误”处理。

4. 输出完整的标识符

为了看到一个完整的标识符, 程序中用字符 ' 将它包围起来。

这个程序的主要特点是用了集合关系运算符 IN, 它大大提高了运算速度。其次, 标识符是逐个字符建立的, 通过 $A[I] := CH$ 来实现。但是, 输出时是一次输出

这个程序是编译程序中实用的读入一个标识符部分的简化形式, 但有某些差异。

许多初学者常常不写如下赋值语句:

LETTER := ['A'..'E'];

DIGIT := ['0'..'9'];

他们认为变量说明可以代替赋值语句, 这是不正确的。

思考题:

分析本题的输入和输出情况。

1. 为什么前两次输出的情况相同?
2. 为什么第四次只输出 ABCD?
3. 为什么后面三次都是标识符错误?

例7-20 已知函数 $F(y) = e^{-y} \cdot \sin(2\pi y)$ 。请编制程序计算并输出其曲线图形。

分析:

这个函数是两个函数的乘积。其中正弦函数是周期性函数。指数函数将随着 y 的增大而衰减。因此, 这个函数的曲线是一个周期性衰减的曲线。

根据终端显示器的特点, 我们让 y 在垂直方向由上而下运动, 用字符 '.' 表示函数值

$x = F(y)$ 在水平方向运动, 用字符 '•' 来表示。

设 y 的增量 DELTA 为 $\frac{1}{16} = 0.0625$;

取函数为两个周期多一点。因此, 曲线选35个点。每行一个点, 取行数 LINE 为35。
曲线每一行为50个字符。即宽度 WIDTH 为50。

y 轴的基准线在第20列的地方。因此, 取 BASE 为20。

由于函数值太小, 无法输出曲线。因此将函数值扩大若干倍。这里取比例因子 RATE 为30。

π 在程序中用 PI 表示。取 PI 为 3.14159。

点号 '.' 表示 y 轴, 它固定在终端显示器屏幕上第20列 (BASE 值) 的位置。

星号 '•' 表示函数值的相对位置。确定方法如下:

1. 计算 $X = F(Y)$;
2. X 值扩大, 乘一个放大倍数 R 。
3. 取整数后加上基数值 (BASE), 即确定在屏幕上的 N 列的位置。
4. 显示该行 N 列为星号 '•', 其余为空白符。
5. 换一行。置 N 列为空白符。实质上是初始化新的一行的第 N 列。
6. 改变 Y 值。
7. 重复上述步骤, 直到35行显示完为止。

程序如 FIG SAR03 PROGRAM 所示。

```
PROGRAM SAR03(OUTPUT);
CONST
  DELTA=0.0625;    PI=3.14159;
  LINE=35;         WIDTH=50;
  RATE=30;         BASE=20;
VAR
  I,J,N:INTEGER;   X,Y:REAL;
  A :ARRAY [0..WIDTH] OF CHAR;
BEGIN
  Y:=0;
  FOR J:=0 TO WIDTH DO A [J] := ' ';
  FOR I:=1 TO LINE DO
    BEGIN
      A [BASE] := '.';
      X:=EXP(-Y)*SIN(2*PI*Y);
      N:=TRUNC(RATE*X)+BASE;
      A [N] := '•';
      FOR J:=0 TO WIDTH DO WRITE(A [J] );
      Writeln;
      A [N] := ' ';
      Y:=Y+DELTA
    END
  END.
```

OUTPUT,

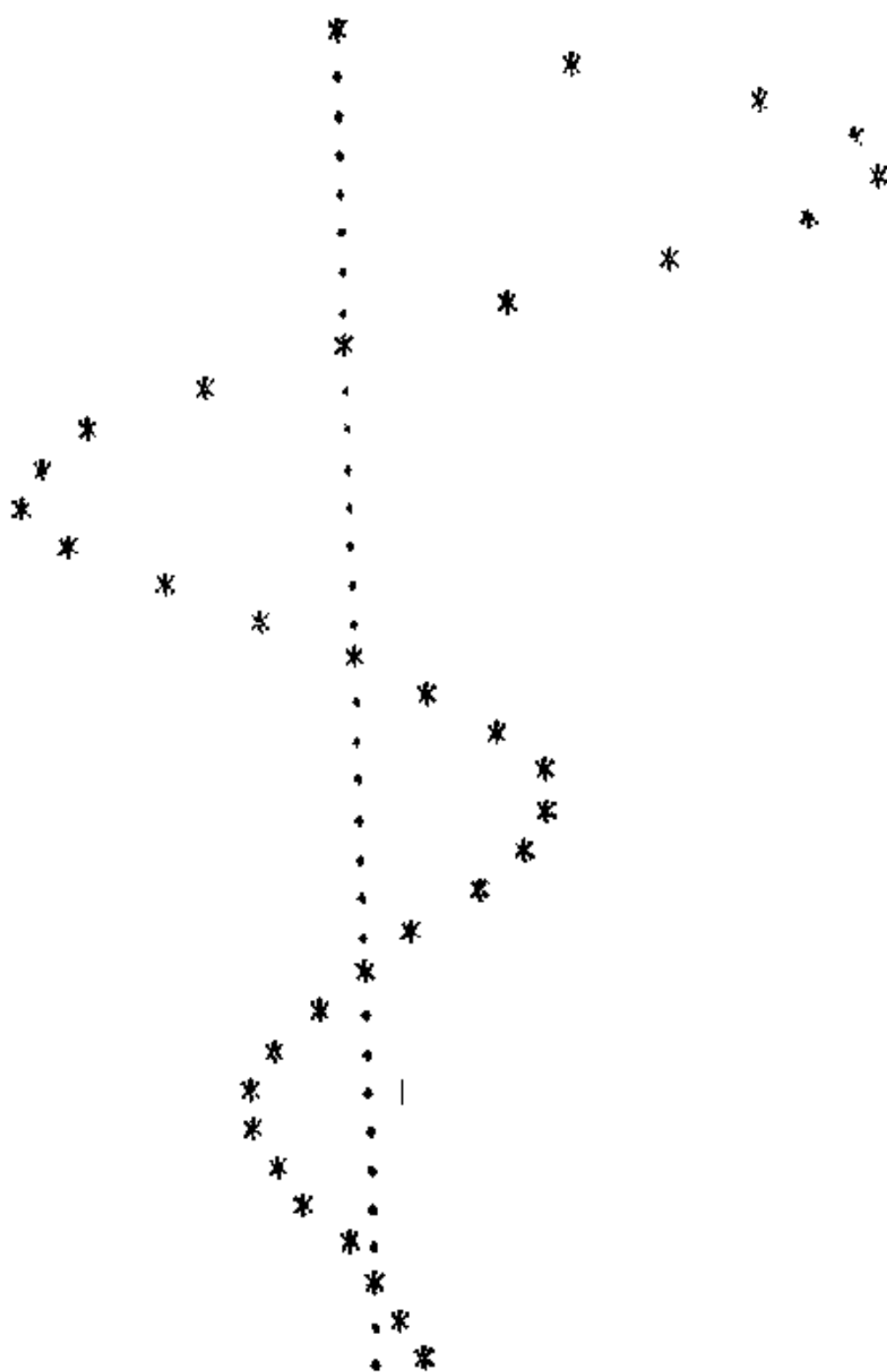


FIG SAR03 PROGRAM

例7-21 建立两个向量。计算其中一个向量的模长度，计算两个向量的内积和外积，输出全部计算结果。

分析：

一个向量包括 X ， Y 和 Z 三个方向，每个方向的长度用实数表示。因此，可以定义一个类型 VECTOR，它有三个元素，元素类型为实数类型。即

TYPE

DIRECTION= (X, Y, Z) ;

VECTOR=ARRAY [DIRECTION] OF REAL;

一个向量的模 (NORM) 为三个分量的平方之和。一个向量的长度 (LENGTH) 为模的平方根。

两个向量的内积是一个实数，两个向量的外积是一个向量。

根据以上分析，对如下变量进行说明：

```

VAR
  S, T : DIRECTION;
  U, V, W : VECTOR;
  NORM, LENGTH : REAL;

```

向量V的三个方向的实数值可以分别表示如下:

$V(X)$, $V(Y)$, $V(Z)$.

一个数组可以作为过程或函数的参数。两个向量的内积是一个实数。计算内积可以用它的基本公式。内积 (INNER PRODUCT) 缩写成 INP.

$INP := U(X) \cdot V(X) + U(Y) \cdot V(Y) + U(Z) \cdot V(Z)$

计算内积还可以定义为一个函数 INP。函数说明如下:

```

FUNCTION INP(U,V:VECTOR) :REAL;
VAR
  R:REAL;
  S:DIRECTION;
BEGIN
  R:=0;
  FOR S:=X TO Z DO  R:=R+U(S) * V(S);
  INR:=R
END;

```

在这个函数说明中, 数值形式参数U和V都是数组类型 VECTOR。函数 INP 的函数值为实数类型。

但是, 必须注意, 一个函数的函数值不允许为数组类型。例如, 两个向量的外积 (OUTER PRODUCT) 是一个向量, 向量属于数组类型。下列函数说明是错误的:

```

FUNCTION OUTP(U,V:VECTOR; VAR W:VECTOR):VECTOR;

```

但是下列过程说明是合法的:

```

PROCEDURE OUTP(U,V:VECTOR; VAR W:VECTOR);
BEGIN
  W(X) := U(Y) * V(Z) - U(Z) * V(Y);
  W(Y) := U(Z) * V(X) - U(X) * V(Z);
  W(Z) := U(X) * V(Y) - U(Y) * V(X);
END;

```

在这个过程说明中, 数值形式参数U和V都是数组类型 VECTOR。变量形式参数W也是数组类型 VECTOR。W为向量U和V的外积。

当然, 用数组作为过程或函数的参数时, 为了节省内存, 用变量参数更好。然而, 这里的数组很小 (只有三个元素), 可以不考虑这个因素, 仍然按照它们各自的功能去选择数值参数或变量参数。

程序如 FIG SAR04 PROGRAM 所示,

```

PROGRAM SAR04(INPUT, OUTPUT);
TYPE
  DIRECTION=(X, Y, Z);

```



```

    VECTOR=ARRAY (DIRECTION) OF REAL,
VAR
    S,      T :DIRECTION,
    U,  U,  W :VECTOR,
    NORM,LENGTH:REAL,
    FUNCTION INP(U,V:VECTOR):REAL,
    VAR
        R:REAL,
        S:DIRECTION,
    BEGIN
        R:=0,
        FOR S:=X TO Z DO
            R:=R+U (S) * V (S) ;
        INP:=R
    END,
    PROCEDURE OUTP(U,V:VECTOR;VAR W:VECTOR),
    BEGIN
        W (X) :=U (Y) * V (Z) -U (Z) * V (Y) ;
        W (Y) :=U (Z) * V (X) -U (X) * V (Z) ;
        W (Z) :=U (X) * V (Y) -U (Y) * V (X) ;
    END,
BEGIN  (• MAIN •)
    FOR S:=X TO Z DO READ(V (S) );
    READLN,
    FOR T:=X TO Z DO READ(U (T) ),
    READLN,
    NORM:=SQR(V (X) )+SQR(V (Y) )+SQR(V (Z) ),
    LENGTH:=SQRT(NORM),
    OUTP(U,V,W),
    WRITELN('NORM(V)=' :11,NORM),
    WRITELN('LENGTH(V)=' :11,LENGTH),
    WRITELN('INP(U,V)=' :11,INP(U,V)),
    WRITELN('OUTP(W (X) )=' ,W (X) ),
    WRITELN('OUTP(W (Y) )=' ,W (Y) ),
    WRITELN('OUTP(W (Z) )=' ,W (Z) ),
END.
INPUT:
    10    5.5    9.9
    2.2    4.5    7.8
OUTPUT:
    NORM(V)=  2.282600E+02
    LENGTH(V)=  1.510818E+01
    INP (U V)=  1.239700E+02
    OUTP(W (X) )=  1.650002E+00

```

```
OUTP(W (Y) )= 5.622000E+01
```

```
OUTP(W (Z) )= -3.290000E+01
```

```
FIG SAR04 PROGRAM
```

在这个程序中，由于函数和过程仅各用一次，为了简化参数，因此将形式参数和实际参数用同一个名称。这样做不会产生错误。此外两个语句 READLN 主要为了从程序设计方面保证输入数据整齐。不用也是允许的。

例7-22 输入两个复数，计算并输出它们的和数及乘积。

分析：

在 PASCAL 语言中，没有复数类型。

为了进行复数运算，可以定义一个记录类型表示复数。这个复数类型包含两个域名，分别表示复数的实部和虚部。域名都属于实数类型。

在定义了复数类型的基础上，按照复数相加及相乘的规律求出其和数及乘积。最后在终端显示器上输出全部计算结束。

程序如 FIG SAR05 PROGRAM 所示。

```
PROGRAM SAR05(INPUT,OUTPUT);
TYPE
  COMPLEX=RECORD
      RE,IM:INTEGER
  END;
VAR
  X,Y,S,P:COMPLEX;
BEGIN
  READLN(X.RE,X.IM);
  READLN(Y.RE,Y.IM);
  WITH S DO
    BEGIN
      RE:=X.RE+Y.RE;
      IM:=X.IM+Y.IM;
      WRITELN('SUM=':8,RE:5,'+I•',IM:5);
    END;
  WITH P DO
    BEGIN
      RE:=X.RE•Y.RE-X.IM•Y.IM;
      IM:=X.RE•Y.IM+X.IM•Y.RE;
      WRITELN('PRODUCT=':8,RE:5,'+I•',IM:5);
    END;
  END.
INPUT:
12    34
56    78
OUTPUT:
      SUM=  68+I•  112
```

PRODUCT=-1980+I*2840

FIG SAR05 PROGRAM

例7-23 建立某个班级的学生情况表。它应包括班号、总人数、每一位同学的学号、姓名、性别、年龄、四门功课（政治、数学、物理、英语）的成绩、总分及平均分。情况表最后要有该班的平均年龄、各门功课的平均成绩及平均分。假定学号从8001开始，人数不超过35人。

分析：

学生的各种情况（元素）不能用同一类型的数据来描述。用数组类型不能完整地表达学生情况，用记录类型来解决这个问题是合适的。

为了输入数据和输出数据的方便，尽量避免使用枚举类型变量。姓名和性别用六个字符的数组类型表示。

各门功课的成绩用整数表示。为了简化域名，用四个整数的数组类型来表示各门功课的成绩。

学生成绩不仅是学生记录情况表中的一项，是记录类型的一个域名，而且本身又是一个记录，它包括各门功课的成绩、总分及平均分，也就是说这个记录又包含三个域名。

班号可以长一些，我们用 ALFA 类型表示。

程序如 FIG SAR 06A PROGRAM 所示。

```
PROGRAM SAR06A(INPUT,OUTPUT),
LABEL
  10;
TYPE
  NAMESEX=ARRAY (1..6) OF CHAR;
  ABARY   =ARRAY (1..4) OF INTEGER;
  CARY    =ARRAY (1..4) OF REAL;
  STUDENT=RECORD
      I, AGE :INTEGER;
      NAME,SEX:NAMESEX;
      SCORE:RECORD
          A :ABARY;
          SUM:INTEGER;
          AVER:REAL
      END
  END;
VAR
  YSUM,TOTAL,N,J:INTEGER;
  YAVER,TAVER :REAL;
  CLASS      :ALFA;
  B:ABARY;    C:CARY;
  K:1..4;     S:STUDENT;
BEGIN
  (• PART 1.....LIST HEAD •)
  WRITELN('FIRST, INPUT CLASS,N !');
```

```

WRITELN('SECOND. INPUT NAME,SEX,AGE. ');
WRITELN('THIRD. INPUT FOUR SCORES. ');
READLN(CLASS,N);
IF N>35
THEN
BEGIN
WRITELN('NUMBER OVER 35. ');
GOTO 10
END;
WRITELN('CLASS:':20,CLASS);
WRITELN('NUMBER:':20,N:6);
FOR J:=1 TO 2 DO WRITELN;
WRITE('NO. NAME SEX AGE ');
WRITE('POLI MATH PHYS ENGL ');
WRITELN(' SUM AVERAGE ');
WRITELN;
( • PART 2.....LIST BODY • )
FOR J:=1 TO N DO
WITH S,SCORE DO
BEGIN
I:=8000+J;
READLN(NAME SEX AGE);
YSUM:=YSUM+AGE;
SUM:=0;
FOR K:=1 TO 4 DO
BEGIN
READ(A (K) );
B (K) :=B (K) +A (K) ;
SUM:=SUM+A (K) ;
END;
READLN;
AVER:=SUM/4;
TOTAL:=TOTAL+SUM;
WRITE(I:4,NAME:7,SEX:6,AGE:4);
FOR K:=1 TO 4 DO
WRITE(A (K) :5);
WRITELN(SUM:6,AVER:7:1)
END;
WRITELN;
( • PART 3.....LIST END • )
WITH S,SCORE DO
BEGIN
YAVER:=YSUM/N;
WRITE('TOTAL AVERAGE:':18,YAVER:4:1);

```

```

        FOR K:=1 TO 4 DO
        BEGIN
            C [K] :=B [K] /N;
            WRITE(C [K] :5:1)
        END;
        TAVER:=TOTAL/(4*N);
        WRITELN(TAVER:11:1)
    END;
10:
    END.
NOTE:
A [1] -B [1] -POL I:POLITICS;
A [2] -B [2] -MATH:MATHEMATICS;
A [3] -B [3] -PHYS:PHYSICS;
A [4] -B [4] -ENGL:ENGLISH;
    FIG SAR06A PROGRAM

```

在这个程序的类型定义部分，类型 STUDENT 是由五个域名（学号，年龄，姓名，性别和成绩）组成的记录类型。其中大域名 SCORE 又是由三个小域名（A，SUM 和 AVER）组成的记录类型，其中小域名 A 表示某个学生每一门功课的成绩。小域名 SUM 表示某个学生的总分。小域名 AVER 表示某个学生的平均分，最后也表示全体学生的平均分。

在这个程序的变量说明部分，主要变量是 S，它属于 STUDENT 类型。YSUM 和 Y-AVER 表示该班年龄之和及平均年龄。TOTAL 和 TAVER 表示该班的总分及总平均分。B 和 C 表示该班某门功课的总分及平均分。CLASS 代表班级。

程序执行分为三个部分：

1. 建立学生情况表“表头”

输入班号及人数。只要人数不超过35人，则建立学生情况表的完整的表头。如果人数超过35人，则作为出错处理。

2. 建立学生情况表“表身”

这一部分主要是一个循环语句。循环语句的成分语句是一个开域语句。主要功能是输入每一个学生各方面的情况，确定学号，计算平均成绩，输出学生情况表的主体部分。同时，统计该班学生年龄之和，每门功课成绩之和，以及全班的总分。

3. 建立学生情况表“表尾”

这一部分主要是一个开域语句。主要功能是计算并输出该班平均年龄，每门功课的平均分及总平均分。

程序后面的注解（NOTE:）表示：

A [1] 和 B [1] 代表某个学生（和累计学生）政治课成绩；A [2] 和 B [2] 代表某个学生（和累计学生）数学课成绩。……

这个程序在运行时，只要输入的数据是合理的，那么就会在终端显示器上输出计算的结果。由于输入数据时也会在终端显示器上回送，因此，用户看到的表格是不理想的。输入和输出混在一起。

为了得到一张完整的学生情况表。用户可以在程序运行时通过行式打印机输出这张表。

这时必须将原来的程序稍作修改, 形成一个新的程序, 如 FIG SAR06B PROGRAM 所示。

```
PROGRAM SAR06B (INPUT,OUTPUT),
LABEL
  10,
TYPE
  NAMESEX=ARRAY (1..6) OF CHAR,
  ABARY   =ARRAY (1..4) OF INTEGER,
  CARY    =ARRAY (1..4) OF REAL,
  STUDENT=RECORD
      I, AGE :INTEGER,
      NAME,SEX:NAMESEX,
      SCORE:RECORD
          A :ABARY,
          SUM:INTEGER,
          AVER:REAL
      END
  END,
VAR
  YSUM, TOTAL, N, J:INTEGER,
  YAVR,TAVR: REAL,
  CLASS      :ALFA,
  B:ABARY,    C:CARY,
  K:1..4,     S:STUDENT,
  P:TEXT,
BEGIN
  REWRITE(P,'LP:'),
  (• PART 1.....LIST HEAD •)
  WRITELN('FIRST. INPUT CLASS,N '),
  WRITELN('SECOND. INPUT NAME,SEX,AGE '),
  WRITELN('THIRD. INPUT FOUR SCORES '),
  READLN(CLASS,N),
  IF N>35
  THEN
    BEGIN
      WRITELN('NUMBER OVER 35!'),
      GOTO 10
    END,
  WRITELN(P,'CLASS:':20,CLASS),
  WRITELN(P,'NUMBER:':21,N:6),
  FOR J:=1 TO 2 DO WRITELN(P),
  WRITE(P,'NO.NAME SEX AGE ');
  WRITE(P,'POLI MATH PHYS ENGL'),
```

```

WRITELN(P,' SUM AVERAGE');
WRITELN(P);
( • PART 2.....LIST BODY • )
  FOR J:=1 TO N DO
    WITH S,SCORE DO
      BEGIN
        I:=8000+J,
        READLN(NAME,SEX,AGE),
        YSUM:=YSUM+AGE,
        SUM:=0,
        FOR K:=1 TO 4 DO
          BEGIN
            READ(A (K) ),
            B (K) :=B (K) +A (K) ,
            SUM:=SUM+A (K) ,
          END,
        READLN,
        AVER:=SUM/4,
        TOTAL:=TOTAL+SUM,
        WRITE(P,I:4,NAME:7,SEX:6,AGE:4),
        FOR K:=1 TO 4 DO
          WRITE(P,A (K) :5),
        WRITELN(P,SUM:6,AVER:7:1)
      END,
    WRITELN(P);
  ( • PART 3.....LIST END • )
  WITH S,SCORE DO
    BEGIN
      YAVER:=YSUM/N,
      WRITE(P,'TOTAL AVERAGE:':18,YAVER:4:1),
      FOR K:=1 TO 4 DO
        BEGIN
          C (K) :=B (K) /N,
          WRITE(P,C (K) :5:1)
        END,
      TAVER:=TOTAL/(4 * N),
      WRITELN(P,TAVER:11:1)
    END,
  CLOSE(P);
10:
  END.
( • NOTE:
  A (1) -B (1) -POLI:POLITICS,
  A (2) -B (2) -MATH:MATHEMATICS,

```

```

A (3) -B (3) -PHYS,PHYSICS,
A (4) -B (4) -ENGL:ENGLISH,  *)
INPUT:
CONTROL801 16
ZHAO FEMALE 21
81 91 91 89
.....
.....
YANG FEMALE 24
81 91 91 81
OUTPUT:

```

```

                                CLASS:CONTROL801
                                NUMBER: 16
NO  NAME  SEX  AGE  POLI  MATH  PHYS  ENGL  SUM  AVERAGE
8001 ZHAO  FEMALE  21   81   91   91   81   344   86.0
8002 QIAN  MALE    22   82   92   92   82   348   87.0
8003 SUN  MALE    23   82   92   92   82   348   87.0
8004 LI   FEMALE  24   84   94   94   84   356   89.0
8005 ZHOU MALE    25   85   95   95   85   360   90.0
8006 WU   MALE    24   94   84   84   94   356   89.0
8007 ZHENG FEMALE  23   93   83   83   93   352   88.0
8008 WANG MALE    22   92   82   82   92   348   87.0
8009 FENG MALE    21   91   81   81   91   344   86.0
8010 CHEN FEMALE  20   90   80   80   90   340   85.0
8011 CHU  MALE    19   89   98   98   70   355   88.8
8012 WEI  MALE    18   80   90   90   80   340   85.0
8013 JIANG FEMALE  21   84   94   94   84   356   89.0
8014 SHEN MALE    22   83   93   93   83   352   88.0
8015 HEN  NALE    23   82   92   92   82   348   87.0
8016 YANG FEMALE  24   81   91   91   84   344   86.0
    TOAL AVERAGE: 22.0  85.8  89.5  89.5  84.6           87.4

```

FIG SAR06B PROGRAM

这个新程序是根据 OMSI PASCAL-1 语言编制的。程序中涉及一些文件类型的概念。主要用到变量 P : TEXT; 用户可以先认可它。有关的详细论述下一章中会讨论。

这个新程序在运行时, 如果输入的数据是合理的, 就会自动在行式打印机上输出一张学生情况表。

通过行式打印机输出的情况如上面所示。

这张学生情况表表示的是控801班 (CONTROL801) 16位同学的情况。这16位同学分别为:

赵 (ZHAO), 钱 (QIAN), 孙 (SUN),
 李 (LI) 周 (ZHOU), 吴 (WU),
 郑 (ZHENG), 王 (WANG), 冯 (FENG),
 陈 (CHEN), 褚 (CHU), 魏 (WEI),

蒋(JIANG), 沈 (SHEN), 韩 (HAN),
杨 (YANG),
性别分别为男 (MALE) 和女 (FEMALE),
平均年龄为: 22.0岁
总平均分为: 87.4分

例7-24 建立某一个教研组的教师情况表。它应包括教师的工作证号, 姓名, 性别, 年龄, 工龄。它还包括教师的简单学位情况, 即有无获得博士、硕士学位, 是何时获得的。若是博士者还要注明获得的地点。最后计算一下该单位的平均年龄, 平均工龄和平均工资。

假定微处理机教研组, 共10人。工作证号从1000开始。

分析:

职工的一般情况用记录类型的固定部分就可以解决。而学位情况必须用记录类型的变体部分来描述。

为了输入数据和输出数据的方便, 尽量避免直接使用枚举类型数据, 而使用字符型数组类型。如姓名和性别用六个字符的数组类型。对于非用枚举类型不可的参数, 则通过输入字符, 然后转换为枚举类型数据, 如域名 STATUS。本程序中对 ALFA 类型数据, 也是通过输入字符然后转换的。

在例7-23中, 由于班级人数不确定, 无法定义一个数组类型来表示全班情况。因而输入和输出数据的过程只能交替进行, 看上去不太清楚。本题的教研组名及人数都已确定, 可以定义一个数组类型来表示全组情况。如定义类型 GROUP, 它的元素类型为记录类型, TEACHER, 共有10个元素。类型 TEACHER 有六个域名: 工作证号 (I), 姓名 (NAME), 性别 (SEX), 年龄 (AGE), 工龄 (SERV), 工资 (PAY) 和学位情况 (STATUS)。其中 NAME 和 SEX 是字符型数组类型。而 STATUS 通过一个变体部分来描述。

程序如 FIG SAR 07A PROGRAM 所示。

```
PROGRAM SAR07A(INPUT,OUTPUT),
LABEL
  10;
CONST
  N=10;
TYPE
  NAMESEX=ARRAY (1..6) OF CHAR;
  DEGREE=(D,M,Q);
  TEACHER=RECORD
      I,AGE,SERV,PAY:INTEGER;
      NAME      SEX:NAMESEX;
      CASE STATUS :DEGREE OF
          D:(YEAR1:INTEGER,
              STATE:ALFA);
          M:(YEAR2:INTEGER);
          Q:(YEAR3:INTEGER);
      END;
  GROUP=ARRAY (1..N) OF TEACHER;
```

```

VAR
  YSUM,SSUM,  PSUM :INTEGER,
  YAVR,SAVER,PAVER:REAL,
  L:1..N;      CH:CHAR,
  G:GROUP,
BEGIN
  (•PART 1.....LIST HEAD•)
  Writeln('GROUP:MICROCOMPT':35);
  Writeln('NUMBER: 10':32);
  Writeln,
  Write('NO. NAME SEX '),
  Write('AGE SERV PAY');
  Writeln(' ( STATUS OF DEGREE ) ');
  Writeln(' ':38,'DEGREE DATE LOCATION'),
  Writeln,
  Write('FIRST. INPUT NAME,SEX,');
  Writeln('AGE,SERV,PAY: ');
  Writeln('SECOND. INPUT STATUS OF DEGREE: '),
  (•PART 2.....LIST BODY •)
  FOR L:=1 TO N DO
    BEGIN
      WITH G (L) DO
        BEGIN
          I:=1000+L,
          READLN(NAME,SEX,AGE,SERV,PAY),
          YSUM:=YSUM+AGE,
          SSUM:=SSUM+SERV,
          PSUM:=PSUM+PAY,
          READ(CH),
          CASE CH OF
            'D':STATUS:=D,
            'M':STATUS:=M,
            'Q':STATUS:=Q,
            ELSE
              BEGIN
                Writeln('STATUS OF DEGREE ERROR: ');
                GOTO 10
              END
            END,
          END,
          CASE STATUS OF
            D:
              BEGIN
                READLN(YEAR1,CH),
                IF CH='F'

```

```

        THEN STATE:='FROMABROAD'
        ELSE STATE:='AT-HOME    ';
    END;
    M :READLN(YEAR2);
    Q:
    END;
    WRITE(I:4,NAME:7,SEX:7);
    WRITE(AGE:6,SERV:6,PAY:6);
    CASE STATUS OF
        D :WRITELN('    PH.D.',YEAR1:6,' ',STATE);
        M :WRITELN('    M.S. ',YEAR2:6);
        Q :WRITELN('    OTHER')
    END
    END;
END;
(• PART 3-----LIST END •)
Y AVER:=YSUM/N;
S AVER:=SSUM/N;
P AVER:=PSUM/N;
WRITE(' AVERAGE:':20);
WRITELN(Y AVER:6:1,S AVER:6:1,P AVER:6:1);
10:
END.
INPUT:
ZHAO MALE 54 32 148
X 1951F
OUTPUT:
STATUS OF DEGREE ERROR;
    FIG SAR07A PROGRAM

```

这个程序的执行部分同样由三大部分组成:

1. 建立教师情况表“表头”

这个表头包括教研组的名称及人数。还有需要统计的项目。

在程序运行时,首先输入某个教师的姓名,性别,年龄,工龄和工资。然后输入该教师的学位情况。

2. 建立教师情况表“表身”

它是通过一个循环语句来实现的,共循环十次。循环语句的成分语句是一个复合语句。它包括如下功能:

- (1) 计算工作证号;
- (2) 输入教师一般情况,进行逐项统计;
- (3) 输入教师学位情况的标志通过情况语句转换为学位情况;
- (4) 根据学位情况确定是否输入其它情况,同样通过情况语句来实现;
- (5) 输出该教师的全部情况。包括一般情况和学位情况。学位情况是由情况语句来实

现。

3. 建立教师情况表“表尾”

这个表尾很简单，只要计算并输出该单位的平均年龄，平均工龄和平均工资。

在运行这个程序时，如果输入的学位情况不符合规定，则会指出“学位情况有错”(STATUS OF DEGREE ERROR₁)。比如输入为：

ZHAO MALE 54 32 148

X 1951 F

输入的第一行内容是正确的，但第二行关于学位情况就错了，字符X，程序不能转换为任何一种学位，于是程序作出错处理。

如果不仅希望在终端显示器上看到这张教师情况表，而且希望把这张表作为一个文件保存在工作盘中。那么将这个程序稍加修改，形成一个新的程序。新程序如 FIG SAR 07B PROGRAM所示。

```
PROGRAM SAR07B (INPUT,OUTPUT),
LABEL
  10;
CONST
  N=10;
TYPE
  NAMESEX=ARRAY [1..6] OF CHAR;
  DEGREE=(D,M,Q);
  TEACHER=RECORD
      I,AGE,SERV,PAY:INTEGER;
      NAME,      SEX:NAMESEX;
      CASE STATUS :DEGREE OF
          D      :(YEAR1:INTEGER,
                   STATE:ALFA);
          M      :(YEAR2:INTEGER);
          Q      :(YEAR3:INTEGER);
      END;
  GROUP=ARRAY [1..N] OF TEACHER;
VAR
  YSUM, SSUM, PSUM :INTEGER;
  YAVR,SAVER,PAVER:REAL;
  L:1..N;      CH:CHAR;
  G:GROUP;     P :TEXT;
BEGIN
  REWRITE(P,'SAR07B','DAT');
  (• PART 1.....LIST HEAD •)
  WRITELN(P,'GROUP:MICROCOMPUTER':35);
  WRITELN(P,'NUMBER:      10':32);
  WRITELN(P);
  WRITE(P,'NO      NAME      SEX      ');
```

```

WRITE(P,' AGE  SERV  PAY'),
Writeln(P,' [STATUS OF DEGREE] ');
Writeln(P,' :38,'DEGREE DATE LOCATION');
Writeln(P);
WRITE('FIRST, INPUT NAME,SEX,');
Writeln('AGE,SERV,PAY, ');
Writeln('SECOND, INPUT STATUS OF DEGREE, ');
( * PART 2.....LIST BODY * )
FOR L:=1 TO N DO
  BEGIN
    WITH G [L] DO
      BEGIN
        I:=1000+L;
        READLN(NAME,SEX,AGE,SERV,PAY);
        YSUM:=YSUM+AGE;
        SSUM:=SSUM+SERV;
        PSUM:=PSUM+PAY;
        READ(CH);
        CASE CH OF
          'D':STATUS:=D,
          'M':STATUS:=M,
          'Q':STATUS:=Q,
          ELSE
            BEGIN
              Writeln(P,'STATUS OF DEGREE ERROR, ');
              GOTO 10
            END
          END,
        CASE STATUS OF
          D:
            BEGIN
              READLN(YEAR1,CH);
              IF CH='F'
                THEN STATE:='FROMABROAD'
                ELSE STATE:='AT-HOME ',
              END,
          M :READLN(YEAR2);
          Q:
            END,
        WRITE(P,1:4,NAME:7,SEX:7);
        WRITE(P,AGE:8,SERV:3,PAY:6);
        CASE STATUS OF
          D :Writeln(P,' PH.D.',YEAR1:6,' ',STATE);
          M :Writeln(P,' M.S.',YEAR2:6);

```

```

      Q :WRITELN(P,'    OTHER')
    END
  END
END;
(•PART 3.....LIST END •)
YAVR:=YSUM/N;
SAVR:=SSUM/N;
PAVR:=PSUM/N;
WRITE(P,'AVERAGE:':20);
WRITELN(P,YAVR:6:1,SAVR:6:1,PAVR:6:1);
CLOSE(P);

```

10:

END.

INPUT:

ZHAO MALE 54 32 148

D 1951F

QIAN FEMALE 49 25 109

D 1981S

.....

.....

SHEN FEMALE 48 24 89

Q

OUTPUT:

(• SAR07B.DAT •)

GROUP:MICROCOMPUTER

NUMBER: 10

NO	NAME	SEX	AGE	SERV	PAY	(STATUS OF DEGREE)	DEGREE DATE	LOCATION
1001	ZHAO	MALE	54	32	148	PH.D.	1951	FROMABROAD
1002	QIAN	FEMALE	49	25	109	PH.D	1981	AT-HOME
1003	SUN	FEMALE	35	11	56	OTHER		
1004	LI	FEMALE	25	3	56	M.S.	1982	
1005	ZHOU	MALE	46	22	78	OTHER		
1006	WU	MALE	44	20	69	OTHER		
1007	CHENG	MALE	40	15	69	M.S.	1982	
1008	WANG	FEMALE	58	37	180	OTHER		
1009	JIANG	MALE	36	15	62	OTHER		
1010	SHEN	FEMALE	48	24	89	OTHER		

AVERAGE: 43.5 20.4 91.6

FIG SAR07B PROGRAM

这个新程序是根据 OMSI PASCAL-1 语言编制的。在执行这个程序时，如果输入的数据符合规定，那么在工作盘中就会自动生成一个新的文件，SAR07B.DAT。文件 SAR07B.DAT 的内容就是微处理教研组十位教师的情况表。这时输入和输出的情况如 FIG SAR07B

PROGRAM 所示。

对照 SAR06B.PAS 和 SAR07B.PAS 可以发现，它们涉及文件部分的差别仅仅在于：

```
REWRITE (P, 'LP,');
```

```
REWRITE (P, 'SAR07B', 'DAT');
```

请读者注意：

如果输入的一个教师是博士，那么输入获得博士学位的时间外，还必须输入获得的地点，中间不允许有空格符分开。

例如，下面的输入是正确的：

```
ZHAO MALE 54 32 148
```

```
D-1951D
```

但是，下面的输入是错误的：

```
ZHAO MALE 54 32 148
```

```
D-1951-F
```

想一想，这是为什么？

例7-25编制一个程序，通过筛选法找出1至10000以内的全部素数，每行8个素数，输出素数表。程序见 FIG SAR08 PROGRAM。

分析：

1. OMSI PASCAL-1 语言版本限制一个集合中的元素最多为64个。因此要解决10000以内素数问题，仅仅用一个集合是不行的。

数组的元素类型可以是任何数据类型，当然也包括集合类型。把筛 S (SIEVE) 和素数表 (PRIMES) 定义为类型 SP，即元素为集合类型的数组类型。

2. 要多少个集合来组成集合类型的数组形式呢？

寻找1至10000以内的素数，考虑到除2以外的全部偶数都不是素数，因此只需要在1至10000之内的全部奇数（包括1在内为5000个奇数）中寻找即可，最后加上素数2。要存放5000个奇数在集合中，则要求 $(5000 \text{ DIV } 64) + 1 = 79$ 个集合。设定集合编号为0..78，由于79个集合可以放 $79 \times 64 = 5056$ 个元素，因此在第79*集合中应该去除56个元素【8..63】。

3. 元素在集合中是如何存放的呢？PDP-11/03系统，一个集合用4个字存放，即 $16 \times 4 = 64$ 位来代表的，即在64位二进制的位置上有1则存放一个元素，是0则不存放元素。如果一个集合存放64个元素，则64位上全部放满1。

用S来记忆（存储）代表5000个奇数的元素，用P来记忆（存储）代表全部素数的元素。

用数组集合S记忆5000个奇数的思路是：在第一个（0号）集合中的0..63分别代表1，3，5，7，9，……127共64个奇数。第二个集合（1号）中的0..63则代表129，131……255共64个奇数。那么第i个集合中的第j个元素代表的奇数是 $(64 * i + j) * 2 + 1$ ，比如第3*集合的第5个元素代表的奇数是 $(64 * 3 + 5) * 2 + 1 = 395$ 。

4. 设立一个由两个整数类型组成的记录 NEXT、ARYE、SETE 分别表示正在处理那个奇数在数组S和P中的下标和相应集合中的元素序号。对应上面的i和j。用开域语句来寻找素数，记录并存放在相应的P中，并记下S的下标和相应集合元素的序号。

5. 具体寻找素数的过程是：

从S中的头一个（第0号）集合的第一个元素3开始。（考虑到1不是素数，从该集合删除。这样奇数的总数为4999个。）因为除1外没有比3更小的奇数了，因此3为素数，找到第一

个素数3以后,就可以从S中删除掉3和3的全部倍数。然后再找第二个,第三个……按此类推,把它们和它们的全部倍数从S中删除,直到S中的全部奇数去掉,全部集合为空集合为止。

6. 输出素数

集合类型数组P中存放着从3至10000的全部素数(3, 5, 7, …)。由于集合类型的数据不能直接通过写语句输出,必须通过某种转换关系或循环语句间接地输出。由于3至9999共有4999个奇数。用循环语句依次检查它们是否在P中,是,则按规律 $(T \cdot 2 + i)$ 输出,每行输出8个素数,形成素数表。

由于2不是通过上述运算求出来的素数,必须另外加上。

给出这个程序主要有三个目的:

1. 用集合类型处理这种问题速度最快,所用内存单元也不多;
2. 当集合的元素超过允许的64个时,可以使用集合数组越过这一限制;
3. 集合中每个元素所代表的真正数值,是程序员约定的,只要按照这种约定使用它,就可以得到正确的结果。

有关程序见 FIG SAR08 PROGRAM.

程序结果见 SAR08 DAT.

```
PROGRAM SAR08(OUTPUT);
CONST
  SETL=63,
  ARYL=78;
TYPE
  SP=ARRAY (0..ARYL) OF SET OF 0..SETL;
VAR
  S, P :SP;
  NEXT:RECORD
    ARYE,SETE:INTEGER
  END;
  J,K,T,C,N:INTEGER;
  EMPTY :BOOLEAN;
  I :TEXT;
BEGIN
  (* PART 1.....INITIALIZE *)
  FOR T:=0 TO ARYL DO
    BEGIN
      S(T) := {0..SETL};
      P(T) := { }
    END;
  S(0) := S(0) - {0};
  S(78) := S(78) - {8..63};
  NEXT.ARYE:=0;
  NEXT.SETE:=1;
  EMPTY:=FALSE;N:=0;
```



```

( • PART 2.....FIND PRIME • )
WITH NEXT DO
  REPEAT
    WHILE NOT(SETL IN S (ARYE) ) DO
      SETL:=SUCC(SETL);
      P (ARYE) :=P (ARYE) + (SETL) ;
      C:=2 • SETL+1;
      J:=SETL; K:=ARYE;
      WHILE K<ARYL DO
        BEGIN
          S (K) :=S (K) - (J) ;
          K:=K+ARYE • 2
          J:=J+C;
          WHILE J>SETL DO
            BEGIN
              K:=K+1;
              J:=J-64
            END
          END;
        IF S (ARYE) = ( )
        THEN
          BEGIN
            EMPTY:=TRUE;
            SETL:=0
          END;
          WHILE EMPTY AND(ARYE<ARYL) DO
            BEGIN
              ARYE:=ARYE+1;
              EMPTY:=S (ARYE) = ( )
            END
          UNTIL EMPTY;
        ( • PART 3.....OUTPUT PRIME • )
        REWRITE(F,'SAR08','DAT');
        WRITELN(F,'      2');
        FOR T:=1 TO 4999 DO
          BEGIN
            J:=T DIV 64;
            K:=T MOD 64;
            IF K IN P (J)
            THEN
              BEGIN
                WRITE(F,(T • 2+1):5);
                N:=N+1;
                IF N MOD 3=0

```

```

        THEN WRITELN(F)
    END
END;
FOR T:=1 TO 2 DO WRITELN(F);
WRITELN(F,' ':5,'THERE ARE',(N+1):5,'PRIMES;');
CLOSE(F)
END.

```

OUTPUT:

```

(•          SAR08.DAT          •)
  2
  3    5    7   11   13   17   19   23
.....
1013 1019 1021 1031 1033 1039 1049 1051
.....
5101 5107 5113 5119 5147 5153 5167 5171
.....
6983 6991 6997 7001 7013 7019 7027 7039
.....
9871 9883 9887 9901 9907 9923 9929 9931
9941 9949 9967 9973
THERE ARE 1229 PRIMES;
FIG SAR08 PROGRAM

```

本章小结

我们回顾一下第六章和第七章，就会发现这些类型说明的共同点，不同点及相互联系。通过编制程序，又会发现不同类型的使用特点。

用户定义的两简单类型和本章讨论的三种构造类型的共同点是必须由用户根据需要按照规定自己确定。主要不同点是，枚举类型和子界类型，只要列出其全部元素或元素的上界和下界，而集合类型、数组类型和记录类型必须指出其元素的类型，因为它是由某些类型构成的新类型，而不是由某些元素构成的类型。枚举类型和子界类型常常是集合类型的元素类型、数组类型的下标类型。枚举型变量和子界型变量又常常是数组型变量的下标。这就是它们之间的主要联系。

三种构造类型的共同点是都必须包含元素类型。元素类型一般不加限制。主要不同点之一，是集合类型中的元素无法直接搜索，而数组类型和记录类型中的元素可以通过下标或域名直接搜索。其次，集合类型和数组类型中的全部元素必须属于同一种类型，记录类型中的全部元素允许属于几种类型，甚至还可以通过变体部分来进行选择。因此，某个集体的建立，可以通过集合类型来实现；建立某个班学生成绩表可以用数组类型（成绩全用实数表示）；建立某个班学生情况表可以用记录类型（姓名，性别用字符数组表示，成绩用实数数组表示）；建立某个单位综合性的情况表必须用记录类型，而且必须有变体部分去解决带选择性的问题。这就是它们之间的主要联系及发展。

枚举类型，集合类型的数据不能通过读语句直接输入，也不能通过写语句直接输

出。布尔类型的数据也不能通过读语句直接输入。因此，凡是可以用其他数据类型（如数组类型、字符类型）代替这些数据类型时，尽量避免使用这些类型。如果一定需要这些类型的数据，则可以通过过程说明去完成输入和输出之间的转换。由于只有整数、实数和字符才能直接输入输出，因此，凡是最基础的元素类型是这些类型时才能直接输入和输出，否则是不行的。这一点在程序设计时必须充分重视。

本章习题

1. 计算并输出 $S = \sum_{i=1}^{10} x_i \cdot y_i$

其中 x_i 的值为 1.05, 2.08, 1.32, 3.56, 0.34,

2.34, 1.38, 9.56, 0.27, 4.12,

y_i 的值为 0.25, 3.42, 1.89, 2.07, 3.14,

1.27, 2.57, 3.14, 4.23, 5.12.

2. 建立矩阵 A (1..10, 1..5), B (1..5, 1..6),
计算并输出两矩阵的乘积矩阵 C.

$$\left(C_{ij} = \sum_{k=1}^5 a_{ik} \cdot b_{kj} \right)$$

3. 用不同的符号同时打印 $y_1 = \sin(x)$, $y_2 = \sin(2x)$, $y_3 = 2\sin(x) + \sin(2x)$ 三条曲线，同时要求打印出坐标。在两条曲线的相交处用符号“#”表示。

4. 数组类型是经常使用的构造型数据类型，但在使用过程中会经常出现数组越界错误 (Array bounds error)。请总结一下这是什么原因造成的，应该采取什么措施加以防止？随意地扩大数组下标范围可能是一个解决的办法，但会带来什么不良后果？请参看第十一章第十个程序中数组设置的技巧。

5. 输入一系列字符，将数字字符，英文字母字符和其它字符分别计算，并求出各占的百分比，输出格式要求如下：

NAME	NUMBER	PERCENT
LETTER	?	? %
DIGIT	?	? %
OTHER	?	? %

遇到‘.’停止计数，用集合类型。

6. 集合类型是重要的构造型数据，它与其它构造类型的根本区别是什么？它可以进行哪些关系运算？它最突出的优点是什么？变量说明中已经定义了集合类型数据，在程序的执行部分还要不要再赋值？OMSI PASCAL-1 语言限制一个集合中的元素最多为 64 个，如果现在有 1000 个，或 10000 个元素要放在集合中，如何解决这个问题？元素在集合中是怎样存放的？

7. 试比较本章例 7-4-1 A 与例 7-4-8 两个程序，后者采用集合类型数据，可以使程序运行的时间缩短为前者的 $\frac{1}{67}$ （两者求解 10000 以内素数的时间之比），请问这是什么道理？请自编一个类似的题目，比较不同方法的程序运行时间。

8. 编一程序把终端上输入的一序列字符中符合标识符定义的部分,变换为 ALFA 类型数据输出显示, 其余内容扔掉不管。

提示: A B C D12, 4E F G H567? # M150F.4.....

A B C D12_____

E F G H567____

M150F_____

9. 编程序建立某班25人的数学课程成绩表, 要求用数组类型和记录类型。其成绩表格式如下:

姓名	性别	平时成绩	期中考试	期终考试	总评成绩
张平	男	90	85	92	?
王良	男	70	62	71	?
....					
李英	女	82	84	75	?
班平均成绩		?	?	?	?

(其中总评成绩=平时成绩×20%+期中考试×30%+期终考试×50%)

第八章 文 件

在 PASCAL 语言里,给出了四种构造类型数据,它们是数组、记录、集合和文件。这四类数据的主要区别集中地表现为访问它们的每一个组成成分(或称元素)时所用的方法不同。对数组来说,它的所有成分都属同一类型,并且只能借助于下标访问它的每一个成分。而对记录来说,它的每一个成分(我们称它们为域),可以是不同类型的数据,并且只能借助于记录名和域名来访问记录的每一个域。而文件,则是相同类型数据的一个序列,我们只能按照每个成分在文件中的自然序列,依次地访问他们,而且每次仅有一个成分是可以直接访问的。在 PASCAL 语言中,文件类型有着它特有的重要性,并且也比较难于掌握与使用,所以我们单列一章,作比较详细的讨论。

在本章,我们将重点介绍如下三项内容:为什么要使用文件,怎样在 PASCAL 语言的程序中说明文件,又应如何对已说明的文件进行操作。最后,再给出几个简单的例证程序。

§1 文 件 概 述

什么是文件?大家都已知道,计算机的操作系统的重要组成部分之一,就是文件管理系统。这里说的文件,一般是指存放在外存储器(如各种磁盘、磁鼓、磁带等)上的一些信息集合,或者干脆就是计算机的某些外设,如终端,读卡机等。在 PASCAL 语言中给出的文件,往往指的也是这些东西,也就是由一系列相同类型数据组成的一个数据序列。这句话概括了文件的两个重要特性:第一,组成一个文件的每个成分,必须是相同类型的数据(这与数组类型有相似之处,但不同于记录类型);第二,文件是一个数据序列。序列,就是按序排列的意思。所以,文件的所有成分之间是有着排列次序的关系的(这与数组结构也差不多)。我们要访问文件的元素,只能按照它们排列的实际次序,一个一个地依次访问它们,而且每次只能直接访问文件的某一元素,访问完这一个,再转向下一个,依次类推。这是文件与数组的最重要的区别,也是对使用文件所加的一个重要限制。我们可以把组成文件的一个基本数据称为文件的元素,或叫文件的成分,或称之文件的记录。最后的这种称呼有时还很普遍,但应注意它与前面讲解过的记录类型数据的记录相区别。为了更清楚起见,我们在下文中一律把组成文件的记录称之为文件的元素,或组成成分。把一个文件的元素个数称为文件的长度。如果一个文件的长度为零,也就是说,这个文件不包含任何元素,则称这个文件为空文件。文件的长度,不是通过文件说明来确定的,它可以随着对文件处理的进展情况,随时加以改变,这又是文件与数组的一个重要区别。

什么情况下要使用文件,为什么必须在程序设计中使用的文件?对此,可以从如下三个方面来回答。第一,一些程序,如 PASCAL 编译程序,要能对使用它的所有用户的任何的 PASCAL 源程序进行编译,而不是只对某一个具体的 PASCAL 源程序进行编译。为此,只能以文件形式,把每一个要被编译的 PASCAL 源程序,提供给编译程序,编译程序也只能以文件的形式,把编译的结果(目标程序清单文件)交付给用户。这就是说,文件是使一个程序可以对不同的输入数据进行加工处理,产生相应的输出结果的一种常用手段。第

二，一个程序装入内存，启动并执行完成之后，就要让它退出内存，以便使得另外的程序可以使用这片内存区域。如果我们把执行过的程序（若在执行过程中，它没有修改自己的代码的话），以文件形式保存在外存中，那么，将来再要执行这个程序时，只要再次把它装入内存加以执行就可以了。这样可以大大节省手工输入时间。与此相联系的，我们也可以用文件形式保留这次下机得到的某些中间结果，以备下次使用。这就是说，文件的使用可以给用户带来许多方便，并提高上机效率。第三，一个程序可能用到数量很大的原始数据，或者产生大量的中间结果，以至于我们没有办法把它们全部装入内存，这时，我们就必须把它们保存在文件中。可以说，使用文件成了摆脱这种困境的最佳手段。

从上面三点说明可以看出，文件的使用是非常重要的。在某些情况下，不用文件就很难解决遇到的困难。

在 PASCAL 语言中，每个文件都被说明为一个变量。但是，它与 PASCAL 语言中的其它变量有着明显的区别。例如，一个程序所操作的文件，可以在这个程序运行之前就存在，还可以在程序执行过之后继续存在，文件的长度可以大于程序本身。所以，必须用某种方法，对 PASCAL 程序通过文件所完成的操作做某些限制。可以说，PASCAL 语言中的文件，是系统中实际文件的一种抽象表示。在程序中，是无法给出表示一个文件实际情形的各种信息的。在多数计算机系统中，必须在用户的 PASCAL 程序首部中的文件参数表中，列出你要使用的全部内部文件的文件名，并在程序的说明部分，再说明每个文件变量，给出该文件元素的类型。在对文件进行任何操作时，都是通过这个内部文件名实现的，而无须过问（有时也根本无法了解到）实际文件（又称外部文件）的具体情况。在内部文件与实际文件之间建立对应关系，是由操作系统中的文件管理系统来完成的。内部文件与实际文件之间的关系，很象我们前面讲解过程与函数时，提到的一个过程的形式参数与调用这个过程时所用的实际参数之间的关系。这种相似性，已在 PASCAL 语言的语法图上清楚地反映出来。在下一节有关地方，我们还会进一步说明这个问题。但必须看到，内部文件并不是实际文件的复制本，这不仅是不必要的（PASCAL 中的文件是只能顺序访问其每个元素的数据结构，因此不必把它的全部元素都放进内存），有时也是无法办到的（例如：一个文件可能比使用它的程序本身还长，甚至于一个文件的长度比整个内存容量还大，无法把整个文件都装入内存）。因此我们说，在任何时候只有文件的一部分元素在内存里。但到底有多少个元素在内存中，又是哪一些元素在内存中，用户不必过问，这些都由每个具体机器的操作系统和所使用的具体的 PASCAL 编译程序决定的。这里我们要强调说明的是，对任何文件在任何时刻都只有一个元素是可以直接访问的。如果一个文件的名为 F，它的可以直接访问的那个元素就用 F'（在 PDP-11/03 系统中用 F^）表示。我们常把 F' 称为 F 文件的指针，或 F 文件的窗口。在访问完文件的一个元素之后，一些对文件进行操作的标淮过程会自动地把文件指针移向文件的下一个元素，这时下一个元素就变成可以直接访问的了。

§2 文件的说明

与 PASCAL 语言的另外三种构造类型数据一样，要使用文件，必须在程序说明部分对它加以说明。一般情况下，要把在一个程序内部使用的每一个文件的文件名，写在程序首部的文件参数表中，例如：

PROGRAM ABC (INPUT, OUTPUT, A, B, C) ; 这个程序要用到五个文件，

它们的名字分别为 INPUT, OUTPUT, A, B 和 C。

接着应在程序的说明部分, 说明这些文件的类型, 更准确地说, 是说明组成每个文件的成分的类型。文件类型说明的一般格式如图8.2-1。

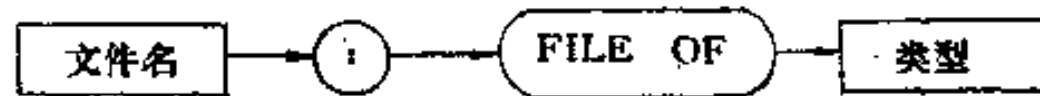


图 8.2-1 文件类型说明的一般格式

上述五个文件的类型说明如下:

VAR

INPUT, OUTPUT : FILE OF CHAR;

A : FILE OF INTEGER;

B : FILE OF REAL;

C : FILE OF RECORD

NEXT : LINK; (假定已在 TYPE 说明中定义了 LINK)

NAME : ALFA;

AGE : 0..100

END;

这里有两点还必须解释一下。

第一, INPUT, OUTPUT 是 PASCAL 语言的保留关键字, 它们有自己特定的含义, 它们是指系统中特定的输入输出文件, 如 PDP-11/03 计算机系统终端, 其它一些计算机系统卡片阅读机和行式打印机等。它们的元素类型只能为字符型, 这是预先约定的。因此, 用户就没有必要再在程序的说明部分来说明它。但在大多数的计算机系统中, 如果在你的程序中要用到这两个文件, INPUT 和 OUTPUT 就必须出现在程序首部的文件参数表中。否则, 就会在编译过程中给出一个错误信息。当然, 如果程序中只用到其中的一个, 则用不到的那一个也可以不出现在这一参数表中。对于运行在 PDP-11/03 计算机系统上的 OMSI PASCAL-1 来说, 程序的首部是任选部分。因此, 程序的文件参数表也就不再有意义, INPUT, OUTPUT 和其它文件名写不写在程序首部中, 对程序运行没有任何影响。

第二, 在说明文件的元素类型时, 如果这个类型本身比较复杂, 或者在说明其它变量时也要用这个类型, 则可以首先在 TYPE 说明中, 定义一个类型, 之后, 在变量说明过程中, 就可以用前面定义过的类型, 来说明文件和其它变量。例如前面的例子中, 也可以用下述办法说明文件 C:

TYPE

.....

CTYPE = RECORD

NEXT : LINK;

NAME : ALFA;

AGE : 0..100

END;

.....

VAR

C: FILE OF CTYPE;

D: CTYPE; (这定义了一个 CTYPE 类型的变量)

这与定义其它构造类型数据的方法是一样的。然而，不管用什么办法说明文件，文件本身都被定义为一个变量，而且自动地引入一个具有成分类型的缓冲区变量，用在文件名后加一个 ↑ (在 PDP-11/03 计算机中用 ^ 代替) 字符表示。如上面例子中，就是 A[↑]、B[↑] 和 C[↑]，它们的类型分别为 INTEGER、REAL 和 CTYPE。这就是我们在上一节中说过的文件的指针。只有处在指针位置的那个成分，才是这个文件中的可以直接访问的成分。

§3 用于对文件进行的标准过程和标准函数

要对文件本身进行读写操作，只能通过调用 PASCAL 语言中给出的标准过程（又可称它们为对文件进行处理的运算符）来完成。为了了解文件的状态（如文件是否为空文件，文件指针是否处在行文文件中一行的末尾，或处在整个文件的末尾等），只有通过调用 PASCAL 的两个标准函数才能知道。下面就逐个地解释一下这些函数与过程。在解释过程中，我们总是假定 F 为用到的文件的文件名。

一、EOF (F) 这是一个标准函数，如果在对文件 F 进行操作的过程中，已经到达了文件中末尾 (END OF FILE)，则它的值为真，否则为假。在对文件进行读写操作时，必须首先了解 EOF (F) 的值，才能保证要进行的操作得到正确的执行。否则，会在程序的运行过程中发生错误。例如，你要从 F 文件中读出它的一个元素，而指针 F[↑] 已移到文件末尾，这就出错了。这是初次使用文件者常犯的一个错误。生活中也有这样的例子，如你有一本只有 100 页的书，你从头一页一页地往后读，到第 100 页就读完了，可你还要再读下一页，就读不到了。在进行写操作时，也有类似的问题。在前面已提到，一个文件的长度是可以变化的，把一个新的元素加到一个原来长度为 n 的文件中，这个文件的长度就变为 n+1。问题是，这个新的元素只能加在文件的末尾，往文件头上或当中什么位置上加的任何企图，都是程序中的错误。我们在下面介绍有关过程时，会反复用到这个概念。

二、EOLN (F) 这是另一个有关的标准函数。有一些文件，如字符文件，是按行 (LINE) 按页 (PAGE) 组织起来的。我们在对它们进行处理时，如果遇到行结束 (END OF LINE) 符时，必须能够识别它。例如，在读一个文件时，遇到了行结束符，可能要做一些相应的处理。我们只能通过调用 EOLN (F) 来判断，碰到的是否是一行的结束符。在写一个 F 文件时，也得有办法把换行符加到 F 文件中去。一般说来，EOLN (F) 只对字符 (由 ASCII 字符组成的) 文件有效。

三、RESET (F) 这是用于把文件指针 (窗口) 复原到文件开始位置的一个标准过程。如果 F 不是一个空文件，则把 F 文件的头一个元素的值传送到指针 F[↑] 中去，并且 EOF (F) 变为假；否则，EOF (F) 为真，F[↑] 的内容无意义。当要从头开始读一个文件时，必须首先调用这个过程。从操作系统的文件管理的角度讲，它起打开一个输入通道的作用。OMSI PASCAL-1 语言还对这一过程的功能进行了扩充，可以用它来建立内部文件与实际文件的对应关系。在前面，我们曾提到这是应由操作系统完成的一项功能。操作系统是怎样找出这两个文件的对应关系的呢？我们简单说明如下。

在一些小型机或微型机的单用户或称单作业 (Single Job) 的系统中，由于整个系统都在一个人控制之下，因此用不着在机器中设置查询用户身分与用机者编号，用户也不必通过

作业控制命令（如作业控制卡片）来说明他将使用的文件。这样，我们就要在 PASCAL 程序内部给出将要使用的实际文件的文件名。在 PDP-11/03 机上运行的 OMSI PASCAL-1 就是这样处理的，其具体办法，是扩展了 RESET 过程的功能，如：

RESET (F, 'A', 'PAS', LEN);

这里 F 是内部文件名，实际文件名为 A.PAS，应该用两个字符串形式给出，即 'A' 和 'PAS'，前一个是文件名，后一个是文件类型，中间用逗号分开，如上所示。

这里的 LEN 是整数型变量参数，在程序执行时，找到了 A.PAS 文件，LEN 中存的是 A.PAS 的长度（以 BLOCK 为单位，每个 BLOCK 存放 512 个字节），若查不到，LEN 为 -1，这样，就可以通过查询 LEN 的值，了解到在盘上是否有这个文件存在。如果 A.PAS 文件不存在，又未用 LEN 检查，这就是程序设计中的严重逻辑错误。

四、REWRITE (F) 这是用于建立一个新的 F 文件的标准过程。调用这个过程时，它首先使 F 变为一个空文件，并使 EOF (F) 变为真。在新建一个文件，或重写一个文件时，必须首先调用它。与 RESET (F) 一样，从文件管理的角度讲，它起着打开一个输出通道的作用。在 PDP-11/03 计算机系统上运行的 OMSI PASCAL-1 语言中，通过把 REWRITE (F) 过程的功能进行扩展来建立内部文件与实际文件的对应关系。例如：

REWRITE (F, 'B', 'LST', LEN);

这里的 F，B.LST 与 LEN 的意义都与上面 RESET 过程中提到的差不多，只是 B.LST 是将在盘上新建的文件的文件名，LEN 的值则表明，在盘上为 B.LST 文件分配盘区时，这个文件可以连续占用的 BLOCK 数量（PDP-11/03 系统规定，任何文件在盘上都得连续存放）。

在多道程序或批处理的系统中，实际文件的文件名是在作业控制命令中给出的，因此，它们不出现在 PASCAL 程序内部。

在这里要提醒大家注意的是，RESET 与 REWRITE 过程不能用于标准输入/输出文件 INPUT 与 OUTPUT。

五、GET (F) GET (F) 这是把文件指针移向 F 文件的下一个元素的一个标准过程，即把 F 的下一个元素的值传送到 F' 中去。这个过程能否正确执行，取决于在调用这个过程时，是否还有下一个元素存在，若有，即 EOF (F) 为假，则可执行，若不存在了，则 EOF (F) 变为真，F' 中的内容不再有意义。当我们要从 F 文件中取出下一个元素时，就要调用这个过程，但必须说明，仅在执行过 RESET (F) 操作之后，才允许进行 GET (F) 操作。

六、PUT (F) 这是把变量缓冲区 F' 中的内容，写到 F 文件末尾位置上，并使文件指针移到文件的下一个位置（此时该位置上并无任何元素，因此，EOF (F) 仍为真）的标准过程。这个过程得以正确执行的条件是：在调用它之前，EOF (F) 的值为真。当我们要把一个新的成分加到文件中去的时候，就要调用这个过程。请大家注意，仅在执行过 REWRITE (F) 操作之后，才允许进行 PUT (F) 操作。

七、CLOSE (F) 在某些机器上使用的 PASCAL 语言，例如在 PDP-11/03 机器上运行的 OMSI PASCAL-1，要用这个过程使实际文件与内部文件“脱离关系”，它包括取消原有的输入或输出通道，取消存储器中的 F 文件的缓冲区，并使输出到磁盘上去的文件成为永久文件。我们在建立一个输出文件的过程中，必须用它把最后剩在缓冲区中的数据传送到实际文件的最后一块中去，并关闭这个文件。否则的话，最后剩在文件缓冲区中的内容，不

会被送到盘上去，已经送到盘上去的内容，也都是以暂时文件的形式存放的，等到程序运行完毕，这个文件就随之失效，用户再也找不到这个文件了。

例 1. 我们要把 $0^\circ \sim 359^\circ$ 之间每一度的正弦值，以文件形式记录在软磁盘上。为此，可用 FIG FIL 01 PROGRAM 所示的程序实现。

```
PROGRAM FIL01 (INPUT, OUTPUT, SNX);
VAR
  X, Y : REAL;
  I, LEN : INTEGER;
  SNX : FILE OF REAL;
BEGIN
  REWRITE (SNX, 'SINX', 'DAT', LEN);
  Y := 3.14159265/180;
  FOR I := 0 TO 359 DO
    BEGIN
      X := I * Y;
      SNX' := SIN (X);
      PUT (SNX)
    END;
  CLOSE (SNX)
END.
FIG    FIL01    PROGRAM
```

在这个例子中

1. 可以不把 INPUT 与 OUTPUT 写在程序的文件参数表中，因为该程序未涉及在终端设备上的入/出操作。

2. SNX 是由实数组成的一个文件，它要出现在程序的文件参数表中，并在程序的说明部分加以说明。这一说明就意味着定义了这个文件的一个变量缓冲区 SNX'，我们正是通过它，往 SNX 文件中写进它的每一个成分的。在这里，请注意 $SNX' := SIN(X)$ 和 PUT (SNX) 两个语句的用法。

3. 在程序的执行部分，通过 REWRITE (SNX, 'SINX', 'DAT', LEN) 来打开文件，使内部文件 SNX 与实际文件 SINX.DAT 对应起来，这是在一些微型或小型计算机系统中常采用的一种办法。这里的 LEN，必须在程序说明部分把它说明为整型变量。当这个 REWRITE 语句执行过后，LEN 中存放的是在用户盘上分配给 SINX.DAT 文件的存贮块 (BLOCK) 数。一般情况下，这个变量用处不大，但当盘上文件很多，或要新建的文件很长时，通过检查这个变量的值，可以防止某些可能出现的错误，如 LEN 的值为 5，用户要建一个长度比 5 BLOCKS 还长的文件，就没法把它写进分配到的盘区里。

4. 最后的 CLOSE (SNX) 语句是很重要的。有了这个语句，程序执行完成以后，就在盘上建了一个永久文件 SINX.DAT，否则，建的是一个暂时文件，程序运行过程中可以使用它，但程序运行一结束，这个文件也就丢失了。

例 2. 我们要对在磁盘上的一个名字为 A.PAS 的文件进行复制，并把得到的副本取名为 B.PAS，同时把 A.PAS 内容在终端上显示出来，则可用下面的程序实现。

```
PROGRAM FIL02A (FIN, FOUT, OUTPUT);
```

```

VAR
  FIN, FOUT : FILE OF CHAR;
BEGIN
  RESET (FIN, 'A', 'PAS');
  REWRITE (FOUT, 'B', 'PAS');
  WHILE NOT EOF (FIN) DO
    BEGIN
      WHILE NOT EOLN (FIN) DO
        BEGIN
          FOUT' := FIN' ;
          PUT (FOUT) ;
          WRITE (FIN' );
          GET (FIN)
        END ;
      READLN (FIN) ;
      WRITELN (FOUT) ;
      WRITELN
    END ;
  CLOSE (FOUT)
END .

      FIG    FIL02A    PROGRAM

```

在这个例子中，必须说明下述几个问题：

1. 因为要在终端上输出，所以，在程序首部中的文件参数表中应有 OUTPUT 这个标准文件名：

2. 在复制一个文件时，必须同时有两个文件，一个是被复制的文件（程序的原始输入数据），一个是复制得到的新文件（程序的输出结果），我们这里分别用 FIN 与 FOUT 表示，它们的元素类型都是 CHAR（注 1，FILE OF CHAR 也可简写为 TEXT）；

请注意这里对 FIN、FOUT 是分别用 RESET、REWRITE 打开的，前者只能读，后者可以写；

3. 与上面一个例子比，这儿多做了两件事。第一，在读输入文件 FIN 之前，总是先检查它是否到了末尾（通过 WHILE NOT EOF (FIN) 语句实现），其理由在介绍 EOF 标准函数时已说明过了。第二，在读每个字符之前，还要通过 WHILE NOT EOLN (FIN) 判断是否到达一行的结尾，还未到就直接读，并把读到的字符写入到输出文件 FOUT 中去，若到了，就要通过 READLN (FIN) 跳到这个文件下一行去读；并用 WRITELN (FOUT) 在输出文件中写进一个换行字符；

4. 在终端上显示 A.PAS 的内容，是通过 WRITE (FIN') 与 WRITELN 实现的。与往 FOUT（即 B.PAS）文件中写相比较，有两点差异。首先，这里没有明确给出标准输出文件的文件名 OUTPUT，这是一种省略的写法，是允许的（请参见本章 §2 开始部分）。其次，这里没有用 PUT^① 而用 WRITE 过程。关于这一点，我们可以这样说，这里的 READ 与 WRITE 分别是 GET 与 PUT 的一种缩写形式，可以简单地把它们的对应关系表示如下：

FOUT':=FIN'

PUT (FOUT)

可以被缩写成: WRITE (FOUT, FIN'),

同理 CH:=FIN'

 GET (FIN)

(这里假定 CH 为已说明过的一个字符型变量) 也可以被缩写成 READ(FIN, CH),

所以, 我们可以把第二个例子改写如 FIG FIL 02 B PROGRAM 所示的程序。

```
PROGRAM FIL 02 B (FIN, FOUT, OUTPUT)
VAR
  FIN, FOUT : FILE OF CHAR;
  CH : CHAR ;
  LEN1, LEN2 : INTEGER ;
BEGIN
  RESET (FIN, 'A', 'PAS') ;
  REWRITE (FOUT, 'B', 'PAS') ;
  WHILE NOT EOF (FIN) DO
    BEGIN
      WHILE NOT EOLN (FIN) DO
        BEGIN
          READ (FIN, CH) ;
          WRITE (FOUT, CH) ;
          WRITE (CH)
        END ;
      READLN (FIN) ;
      WRITELN (FOUT) ;
      WRITELN
    END ;
  CLOSE (FOUT)
END .
```

FIG FIL 02 B PROGRAM

在这个程序中, 到底采用哪种形式, 没有多大差别, 用 GEN, PUT 也许可以少用一个变量单元 (读来的与写出的字符都通过 FIN' 与 FOUT' 处理), 但 READ 与 WRITE, 在多数编译程序中, 功能更强, 尤其在对 TEXT 类型文件进行处理时更是如此。这些内容还将在下一节详细说明。

5. 程序在最后用了 CLOSE (FOUT) 语句关闭 FOUT 文件, 这是必须的。但对 FIN 文件, 我们并未进行关闭操作。必须说明, 如果要反复读 FIN 文件, 最好在读完一遍之后, 使用 CLOSE (FIN) 语句关闭它, 以便撤消它占用的一个输入通道, 在开始下一次读之前再用 RESET (FIN) 重新打开, 否则, 每读一遍都要进行 RESET (FIN), 而不用 CLOSE (FIN), 则每读一遍都要占用一个输入通道, 多次之后, 把可用的输入通道都占用了, 再要读, 就没有通道可用了。这是一个错误。系统会指出打开的文件太多, 并停止运行这一程序。如果只读一遍, 有无 CLOSE (FIN) 语句, 对程序运行无任何影响, 所以常常可以省略。

§4 TEXT 文件

TEXT 文件，又称为文本文件，或行文文件。它是由字符组成的，其定义如下：

TYPE TEXT = FILE OF CHAR;

文本文件是各种类型的文件中最常用的一种。因为大部分的输入/输出操作都是以字符形式进行的。因此，怎样才能最方便地使用这类文件，是必须解决的一个问题。在这一节，准备就下述四个方面进行讨论：

- 一、有关 INPUT 与 OUTPUT 文件使用过程中的几点说明；
- 二、TEXT 文件入/出操作过程中，自动的数据类型变换；
- 三、TEXT 文件中的换行，换页与文件结束处理；
- 四、应用 TEXT 文件时，READ 与 WRITE 标准过程中的参数处理；

一、INPUT 与 OUTPUT 文件 在前面，已多次提到标准的输入/输出文件 INPUT 与 OUTPUT，它们所对应的具体设备，在不同的系统中，可以不一样。但它们的成分只能是那些可打印的字符与少数控制字符，即 INPUT 与 OUTPUT，只能是文本文件。我们已约定如下：

```
TYPE
    TEXT=PACKED FILE OF CHAR;
VAR
    INPUT, OUTPUT, TEXT;
```

这是大家约定好的事情，所以在用户程序中，无须再进行上述说明，这一点在 §2 开始部分已经提到过。。我们在第三节中还提到过，RESET 与 REWRITE 语句不能用于 INPUT 与 OUTPUT 文件。

在编制一个程序时，往往总伴有某些入/出操作，也就是说，在我们的程序中，会频繁地使用 INPUT 与 OUTPUT 文件，如果每逢用到它们，都得给出文件名，那就太麻烦了。为了简便，常常在 READ 与 WRITE 语句中把 INPUT 与 OUTPUT 文件名省掉不写。也就是：

READ (INPUT, CH)	简写成 READ (CH);
WRITE (OUTPUT, CH)	简写成 WRITE (CH);
READLN (INPUT)	简写成 READLN;
WRITELN (OUTPUT)	简写成 WRITELN;
EOF (INPUT)	简写成 EOF;
EOLN (INPUT)	简写成 EOLN;

实际上，我们已约定，如果在 READ 与 READLN 语句中未直接给出文件名，则使用的就是 INPUT 文件；在 WRITE 与 WRITELN 语句中未直接给出文件名，则使用的就是 OUTPUT 文件。这些规则，在前面几节中早就用到了，在这里只是再重复说明一遍。

当然，在 INPUT 与 OUTPUT 之外还可以应用另外一些 TEXT 文件。这时，这些文件名不仅要出现在程序首部的文件参数表中，而且应在程序的说明部分，说明其成分的类型。在程序的执行部分用 RESET 或 REWRITE 语句打开它们。

二、TEXT 文件入/出操作过程中自动的数据类型变换 在前面已经说过，在 PASCAL

语言中，我们只能依次地访问文件的每一个成分，而且每次只能访问文件的一个成分。这里又说到，INPUT 与 OUTPUT 是文本文件，即它们的成分为字符。如果这两个说法都成立，那么，用 READ 与 WRITE 语句实现的入/出操作，就只能按字符形式进行了。而实际情形并不是这样。如果 I 为整型变量，A 为 ALFA 类型变量，R 为实型变量，我们可以用 READ(I)，READ(R) 与 READ(A) 分别从输入设备上读来一个整数，一个实数和一个长度为 10 的字符串。这里的三个 READ 语句，都不止读来一个字符，而且也不把读来的若干个字符的 ASCII 码简单地作为结果存放到 I，R 与 A 中去，而是把他们分别地变换为与 I，R，A 相应的类型。这就是这里说的 TEXT 文件入/出过程中自动的数据类型变换。例如，在执行 READ(I) 时，我们通过终端打入两个字符‘5’，这是说，我们要把 10 进制数的 55 送到 I 单元中去，而字符‘5’的 ASCII 码为 00110101，如果把两个字符‘5’的 ASCII 码直接送到 I 单元，则为 0011010100110101，这并不是 10 进制数 55 的二进制码。因此，不能这样送，而应把它变成 0000,0000,0011,0111 才对。实际上，机器执行 READ(I) 语句时，确实自动地把打进去的两个字符‘5’，变换成十进制数 55 的二进制形式，再存放到 I 单元去的。在执行 READ(R) 与 READ(A) 的过程中也伴有相应处理。反过来，当机器执行 WRITE(I)，WRITE(R) 等输出语句时，也要把存放在内存中的、用二进制形式表示的 I 与 R 的值，再变换为字符形式加以输出。这种入/出过程中的自动的数据类型变换，对于简化用户程序设计，提高效率是十分重要的。设想一下，如果机器不自动地完成这一变换，那么，当用户要读入一个整数类型变量时，他就必须写一段程序，来实现这一变换，这不仅增加了程序设计的工作量，而且还大大增加了程序设计的难度。为了让读者更好地了解这一自动变换的含义，我们给出可以完成这一变换的两个例证性的小程序。

第一个程序，把键盘打入的相应字符变成 10 进制数的二进制码。其程序如 FIG FIL03 PROGRAM 所示。

```

PROGRAM FIL03(INPUT,OUTPUT);
VAR
  I : INTEGER ;
  CH : CHAR;
  DIGITS : SET OF '0'..'9' ;
BEGIN
  DIGITS := ['0'..'9'] ;
  I := 0 ;
  READ(CH) ;
  WHILE CH = ' ' DO READ(CH) ;
  IF CH IN DIGITS
  THEN
    BEGIN
      REPEAT
        WRITELN(I) ;
        I := I * 10 + ORD(CH) - ORD('0') ;
        READ(CH)
      UNTIL NOT ( CH IN DIGITS ) ;
      WRITELN(I)
    
```

```

        END
    ELSE WRITELN('BAD INTEGER')
END .
INPUT:          OUTPUT:
12345           0
                1
                12
                123
                1234
                12345
W              BAD INTEGER

```

FIG FIL03 PROGRAM

第二个程序，把键盘打入的若干个字符形成一个 ALFA 类型数据。程序如 FIG FIL04 PROGRAM 所示。

```

PROGRAM FIL04(INPUT,OUTPUT);
VAR
    I,J : INTEGER ;
    CH : CHAR ;
    LETTERS : SET OF 'A'..'Z' ;
    DIGITS : SET OF '0'..'9' ;
    A : ALFA ;
BEGIN
    LETTERS:= ['A'..'Z'] ;
    DIGITS:= ['0'..'9'] ;
    I:=0; J:=10;
    READ(CH); WHILE CH=' ' DO READ(CH);
    IF CH IN LETTERS
    THEN
        BEGIN
            REPEAT
                IF I<J
                THEN
                    BEGIN
                        I:=I+1; A [I] :=CH
                    END;
                READ(CH)
            UNTIL NOT (CH IN LETTERS) AND NOT (CH IN DIGITS);
            IF I<J
            THEN
                FOR J:=10 DOWNTO I+1 DO A [J] :=' ' ;
            WRITELN (A)
        END
    ELSE WRITELN('BAD ALFA TYPE')

```


END.	
INPUT:	OUTPUT:
ALFA	A. FA
UNIVERSITY	UNIVERSITY
ABCDEFGHIJKL	ABCDEFGHIJ
5WORDS	BAD ALFA TYPE

FIG. FIL04 PROGRAM

对这两个程序简单说明如下:

在第一个程序中:

1. BEGIN 之后的第二行是为了取消打入整数之前所打入的全部空格字符(SPACE), 直至遇到不是空格字符为止。

2. IF 语句首先判断取来的字符是否为数字符。即是否属于 '0'..'9'。如果不是, 则输入的不能形成整数。所以输出语句输出 BAD INTEGER 指示信息。如果是数字符, 则可用 $\text{ORD}(\text{CH}) - \text{ORD}('0')$ 求出这个数字符所对应的十进制数的二进制码。如字符 '5' 的 ASCII 码为 00110101, 而字符 '0' 的 ASCII 码为 00110000, 二者相减为 00000101, 是数字 5 的二进制码。前一次求得的值与这一次求得的值是什么关系呢? 前次值乘以十再加上这次求得的值, 就是两个字符的相应十进制表示的数值。例如, 前一次求得的值为 a, 这一次求得的值为 b, 下一次求得的值为 c, 则该数值应为

$$(a \cdot 1010 + b) \cdot 1010 + c$$

这就是向 I 赋值那个语句的含义, 上式中的 1010 为整数十的二进制码。

接下去一个 READ(CH) 是为了读入下一个字符, 直到读来的字符不再是数字符为止, 这表明要输入的数已输入完了。这是通过 UNTIL NOT (CH IN DIGITS) 来控制的。第一个 WRITE LN(I) 语句是为了显示 I 的初值 (程序开始时设置它为零) 和中间结果。第二个输出语句输出最后结果。读者上机做该题时, 会发现输入和输出的字符完全一样。不过, 请注意: 输出并不是把输入的字符简单地送出来, 而是经过变换的。输入的字符变为二进制数, 是我们的程序完成的; 而二进制数变成字符输出, 是机器自动完成的。

3. 这是个例证程序。机器中真正实现起来还要复杂一些。如先得检查变量类型, 还要检查输入的数是否太大造成溢出, 或处理完这个变量 I, 可能还有别的变量要输入等。这些都没做, 以求程序简单易懂。

在第二个程序中:

1. 先要回顾一下 ALFA 类型。通常, ALFA 为 ARRAY(1..10) OF CHAR 的缩写形式。ALFA 类型主要用以定义标识符 (IDENTIFIERS), 而标识符又要求: 第一个字符为字母, 其余的字符可以是字母或数字, 其中不得有专用字符 (包括空格)。我们的程序是做了这种检查的。

2. ALFA 类型数据的长度为 10, 但我们可能只想用一个字母或再跟几个字符来给出一个标识符。这时, 不足 10 个的那些位上要填上空格。然而, 如果标识符的长度超过 10, 超过的那些字符一律无效。但这种用法不能说是错误的。程序将超过的部分也读进去, 然后“扔掉”就是了。

3. 在读一个 ALFA 类型的数据时, 当遇到即不属于字母, 又不属于数字的字符时, 便结束输入。程序里是用 REPEAT 语句中的 UNTIL 控制的;

4. 程序中的 WRITE(A)语句输出最终结果。可以比较一下,在输入不足10个,恰好10个和10个以上字母与数字字符时,输出结果之差异;

5. 最后一个输出语句用以指出错误。当遇到的第一个非空格字符不是字母时,表示输入有错;

6. 在这里,对 ALFA 类型数据的读入由我们的程序处理,而输出则由计算机自动处理。我们也可以把读入处理也留给机器,这时程序就要作相应处理。

在做了上述说明后,读者一定能够看懂这两个程序。从而对 TEXT 文件入/出操作中自动的数据类型变换有些实际体会。

在上面给出的例子中,选用的只是 INPUT 与 OUTPUT 标准的输入输出文件。实际上,这种自动类型变换适用于所有 TEXT 类型的文件。TEXT 文件便于阅读,作为人一机之间相互通讯是十分方便的。但在另外一些情况中,就不宜使用 TEXT 文件。这一点我们在稍后一点的地方介绍,并给出有关例证程序。

三、TEXT 文件中的换行、换页,与文件结束的处理 通常,我们总要把一个 TEXT 文件,如一个程序或一批数据分成许多行,或者还可以分成几页,而不是简单的把它组织成一长串字符。在不同的计算机和不同的操作系统上,表示这种行、页结构的方法也是不同的。表示一行或一页的结束,最简便的方法是用特定的控制字符。如用 ASCII 字符集中的 CR (回车)与 LF (换行)两个控制字符表示行结束;用 FF (换页)表示页结束。在 PDP-11/03 计算机系统中就是采用这种办法的。若在所用的计算机输入输出装置的字符集里,没有这样的控制字符,这就意味着必须用其它方法来表示行与页的结束了。但是,不管用什么方法表示它们,在 PASCAL 语言中,必定向用户提供出某些标准过程与标准函数,来建立具有行、页结构的文件,识别输入文件中的行结束、页结束与整个文件的结束。

下面的讨论,将用在 PDP-11/03 系统上运行的 OMSI PASCAL-1 为例来进行。

页结束,换页等问题,不很常用,多数编译程序也不处理这件事,我们暂不予讨论。

关于行结束,我们是通过调用标准函数 EOLN(F) (这里 F 为一个 TEXT 文件的名字)来判断的。当 F 文件的指针遇到行结束符时, EOLN(F) 为真, F⁺ 的内容为空格 (SPACE); 否则的话, EOLN(F) 为假, F⁺ 存放的是 F 文件当前成分的内容。

为什么需要判一行的结束呢?这是因为行结束符是这一行与下一行之间的分隔符。在读完一行之后,也就是遇到本行的结束符的时候,可能要做一些相应处理,然后再读下一行。这时只能通过调用 EOLN(F) 这个标准函数才可以了解到是否到了一行的末尾。在 EOLN(F) 为真时,表示已遇到行结束符,此时 F⁺ 的内容为空格字符。之后就是跳到下一行去读,这时应该用 READLN(F) 语句来表达我们这一意图。它表示跳过本行的结束符,而把下一行中的头一个字符取入 F⁺ 中。这里的 READLN(F) 相当于如下语句:

```
IF EOLN(F) THEN GET(F);
```

如果要读出完整的一行,之后转去读下行的头一个字符,则可用:

```
WHILE NOT EOLN DO GET(F);
```

```
GET(F)
```

也可以写成为:

```
WHILE NOT EOLN(F) DO READ (F, CH);
```

```
READLN(F)
```

假定这里的 CH 已在程序说明部分被说明为字符变量。READLN 是一个结束读本行,跳

到下一行头一个字符上去的标准过程。

当然, READLN 也可以用于跳过本行中尚未读过的那些文件成分,“直接”跳到下一行的头一个字符上去。准确地说,这是程序员的意图,或者说程序员所看到的效果。系统还是一个成分一个成分的读文件,并判别读来的成分是否为行结束。若不是,继续取下一个成分,并且不对取来的这些元素的内容进行处理与应用。这在实际中是常常应用的办法。用户要处理好“跳到”下一行的时机,保证有用的成分读全,扔掉的确实是那些不用的成分。

WRITELN(F) 是另外一个标准过程,它结束当前正在输出的一行,并开始新的一行,也即在当前一行的结束处加 CR 与 LF (有时也可以认为 LF 在下一行行首。但请注意:它并非是下一行的头一个字符,因为它只是一个控制字符)。这个操作是必要的,因为只有经过它,才能把建立的文件按照用户需要分成许多行。以便在终端上显示或行式打印机上输出。否则,就只能把一个文件都输出在唯一的一行上,这会超出终端或行式打印机一行所允许的字符个数(80或132个字符)。超出的部分是看不到的。

在 READLN 和 WRITELN 中也可以没有文件参数名,如前所述,这是对标准入/出文件 INPUT 与 OUTPUT 分别进行换行读与换行写的情形。

关于 TEXT 文件的文件结束,是用控制字符 CTRL/Z (ASCII 码的第 26 个字符) 表示的。判断文件结束 (END OF FILE) 是通过调用标准过程 EOF(F) 来完成的。

四、在应用 TEXT 文件时,对 READ 与 WRITE 标准过程中的参数处理 当我们用 READ 与 WRITE 语句对 TEXT 文件进行读写操作时,在 READ 或 WRITE 语句中可以同时出现若干个变量参数。这些参数的类型只能是字符型、整数型和实数型,在某些机器上,还允许字符数组型。对于 WRITE 语句,还可以为布尔型。请看下面一个例子,程序如图 FIG FIL05 PROGRAM 所示。

```
PROGRAM FIL05(INPUT,OUTPUT);
VAR
  INPT,OUTPT: TEXT;
  I,J: INTEGER;   R1,R2: REAL;
  CH1,CH2: CHAR;  B1,B2: BOOLEAN;
BEGIN
  RESET(INPT,'A','DAT');
  REWRITE(OUTPT,'B','DAT');
  B1 := TRUE; B2 := FALSE;
  READ(INPT,I,J,CH1, R1,CH2,R2);
  WRITELN(OUTPT,R2, CH2,R1:5:3,CH1,J:5,I:5,CH2,B1,B2,R1:5:3);
  WRITELN(I:5,J:5,R1:5:3,R2,CH1,CH2,B1,B2);
  CLOSE(OUTPT)
END
(*          A.DAT
  12 543X 1.325R 12578.4          *)
OUTPUT:
  12 543 1.325 1.257840E+04XR TRUE FALSE
(*          B.DAT
  1.257840E+04R 1.325X 543 12R TRUE FALSE 1.325          *)
FIG FIL05 PROGRAM
```

这里要说明三点。

1. WRITE 和 READ 过程中可以有多个参数, 这些参数的类型可以是不同的。但应注意, 在读的过程中要判文件结束, 行结束等 (这里从略)。

2. 在读过程中, 是按这些参数从左到右的次序逐个处理的。处理完一个, 接着处理下一个, 直到全部处理完毕。因此, 上面的 READ 语句也可以写成如下形式:

```
READ(INPT, I);  
READ(INPT, J);  
READ(INPT, CH1);      或者      READ(INPT, I, J, CH1);  
READ(INPT, R1);  
READ(INPT, CH2);      或者      READ(INPT, R1);  
READ(INPT, R2);      READ(INPT, CH2, R2);
```

对 WRITE 过程也一样。

值得注意的是: 出现在 READ 过程中的变量次序, 必须与 INPT 文件中数据存放次序一致。就是说, 在 INPT 文件中, 前面有两个整数, 接着是一个字符, 后面是一个实数, 下面又是一个字符, 再下面又是个实数。否则, 可能出现数据类型不相容的错误, 如 INPT 文件 (实际上即指 A.DAT 文件) 中的第一个字符为 A, 则一读到它, 发现它不是数字符, 与 READ(INPT, I) 中的 I 的类型相矛盾, 机器会指出 "BAD INTEGER", 进一步说, 如果我们这次按某一次序把不同类型变量写入 OUTPT 文件, 下一次再读时, 仍需按原来次序读出。否则, 不是产生读过程中的变量类型不相容的错误, 就是读出来的内容毫无意义 (注意: B1, B2 可以写入 OUTPT 文件, 但不能用 READ 语句去读它们, 因为 READ 语句不允许有布尔型参数)。

3. 在读写过程中, 变量的自动类型转换总是进行的。

4. 如果在上面的 READ 与 WRITE 语句中, 不给出文件名, 则入/出将在指定的入/出设备上进行。

5. 如果把上面的 READ 与 WRITE 语句换成 READLN 与 WRITELN, 那就表示: 全部读入的内容都要从某一行上读来; 全部输出的内容都要在某一行内写出。完成这些入/出后, 不管本行还剩多少内容, 都要使文件指针移到下一行的第一个字符上去。对于 READ 语句, 相当于把 READ(INPT, R2) 变成 READLN(INPT, R2)。或者说, 在 READ(INPT, R2) 之后, 再增加一个 READLN(INPT) 语句。对于 WRITE 语句也一样, 把要输出的全部内容都输出完毕, 再执行一个回车、换行操作。即把要写的全部内容都写到 OUTPT 文件上之后, 再在该文件中写入 CR 与 LF 两个控制字符。

在结束 TEXT 文件的介绍之前, 还要强调一下:

1. 分行与分页的结构, 一般只适用于 TEXT 文件;

2. OMSI PASCAL-1 版本中, 未给出一种完全可行的用以标记与识别非 TEXT 类型的文件的结束的办法。这就要求用户自己想办法加以解决。例如, 通过一个用户一定不会用到的整数, 来标记与识别一个整型文件的结束。或以文件中元素个数来控制文件的结束等。总之, 用户不能用 EOF(F) 这个标准过程去判断非 TEXT 文件的结束。

§5 几个简单的例证程序

为了掌握文件的概念及其应用，下面列举六个例证程序。

例8-1 计算 $0^\circ \sim 89^\circ$ 的每一度的正弦值，并将结果建立一个文件。在行打印机或终端显示器上输出。而且以 SINX.DAT 作为文件名，将结果建立在用户盘上。程序见图 FIG FILE3 PROGRAM

```
PROGRAM FILE3(OUTPUT,F1,F2) ;
VAR
  I : INTEGER ;
  X,Y : REAL ;
  F1 : TEXT ;
  F2 : FILE OF REAL ;
BEGIN
  REWRITE(F1,'LP:') ;
  REWRITE(F2,'SINX','DAT') ;
  X := 3.1415926/180 ;
  FOR I := 1 TO 90 DO
    BEGIN
      Y := SIN( (I-1)*X) ;
      F2* := Y ;
      WRITE(Y:12:8) ;
      WRITE(F1,Y:12:8) ;
      PUT(F2) ;
      IF (I MOD 5 = 0) OR (I > 39)
      THEN
        BEGIN
          WRITELN ;
          WRITELN(F1)
        END
      END
    END ;
  CLOSE(F1) ;
  CLOSE(F2)
END
```

OUTPUT: (ON TT: AND LP:)

0.00000000	0.01745241	0.03489950	0.05233595	0.06975647
0.08715573	0.10452850	0.12186930	0.13917310	0.15643450

.....

0.99619460	0.99756400	0.99862950	0.99939080	0.99984770
------------	------------	------------	------------	------------

(• SINX.DAT

000/ 000000	000000	036616	174131	037016	171306	037126	057072
-------------	--------	--------	--------	--------	--------	--------	--------

020/ 037216	156173	037262	077265	037326	011405	037371	113242
-------------	--------	--------	--------	--------	--------	--------	--------

.....

FIG FIL06 PROGRAM

结合该程序，说明如下四点：

1. 我们可以把某些设备看作文件，如行式打印机、终端显示器等。行打机的设备名为“LP:”，终端显示器的设备名为“TT:”。然而终端是我们所用的 PASCAL 语言默认的入/出设备。因而在入/出语句中可以省略这一设备名。在这种情况下，不能对其执行 RESET 和 REWRITE 操作。如果非要执行这种操作，则设备名“TT:”不能省去。

2. 程序中说明了两个文件 F_1 和 F_2 。 F_1 代表 LP:， F_2 代表用户盘上的 SINX.DAT。两个写语句分别将每一度的正弦值送到终端、行式打印机；一个 PUT 语句则把 Y 的值送到用户盘上去。这样做的目的，是让读者看到应如何去使用各种设备。

3. 三个输出语句后面的 IF 语句是为了处理换行问题。在每行上输出五个正弦值之后就换行；或者在结果输出的最末一行中，不管本行是否已满五个正弦值，也换行。这与前面讲过的输出语句一样，换行也使用了两个语句。

4. 必须用 CLOSE 语句关闭 F_1 与 F_2 两个文件，否则是不行的。不关闭 F_1 ，问题还不大，但无最后一个 WRITELN(F_1) 语句，则最后一行结果不在行式打印机上输出；不关闭 F_2 ，程序结束后在用户盘上找不到 SINX.DAT 文件。

5. 建在盘上的文件 SINX.DAT 不是 TEXT 文件。因为程序中是用 PUT(F_2) 语句把正弦值写进 SINX.DAT 中，未经类型变换，因此，不能按通常的办法来显示或打印 SINX.DAT 文件。这就可以看出 PUT 和 WRITE 语句的区别了。

例8-2 把一个放在用户盘上的文件 FIL07.DAT，反复地读出在终端加以显示。该文件是用编辑程序建立的。它由字符 * 排列成六个汉字，“同志们欢迎你”。文件长度不超过40行。连续显示它，就可以取得如同放映幻灯片一样的效果。程序如 FIG FIL07A PROGRAM 所示。

```

PROGRAM FIL07A(OUTPUT,F) ;
LABEL
  1 ;
VAR
  F : TEXT ;
BEGIN
  ;
  RESET(F,'FIL07','DAT') ;
  WHILE NOT EOF(F) DO
    BEGIN
      WHILE NOT EOLN(F) DO
        BEGIN
          WRITE(F') ;
          GET(F)
        END ;
      Writeln
    END ;
  CLOSE(F) ;

```

```

      GOTO 1
END .
      FIG FIL07A PROGRAM

```

这个程序的功能，类似于 RT-11 操作系统中的键盘命令：

```
TYPE/COPIES:n FIL07.DAT
```

其中 n 是一个比较大的正整数。

结合该程序，说明如下两点：

1. 要重复显示文件内容，就要到磁盘上反复地去读它。每读一次，都要首先执行一次 RESET 语句。但是，不用特殊办法处理，系统是不允许执行过多次数的 RESET 语句的。为此，每读完一次，程序都执行一次 CLOSE 语句，以关闭由 RESET 语句打开的一个输入通道。这样，就可以无限制的执行 RESET 语句，从而可以反复地读相应文件了。这里是用 GOTO 语句实现这一反复操作的。

2. 在一个程序中，重复地读磁盘上的相同内容，不仅速度太慢，而且又很磨损设备。应尽量避免。例如可以首先将文件内容读到程序内部的一个数组中，用时直接从这一数组中取出来，而不必再到磁盘上去寻找（此法的前提是：该文件不十分长，我们的程序中可以容得下它）。为此，将这一程序改写为 FIG FIL07B PROGRAM。

```

      PROGRAM FIL07B(OUTPUT,F) ;
VAR
      F : TEXT ;
      I,J : INTEGER ;
      A : ARRAY (1..80,1..36) OF CHAR ;
BEGIN
      RESET(F, 'FIL07' , 'DAT') ;
      IF EOF(F)
      THEN WRITELN( 'FIL07.DAT IS AN EMPTY FILE')
      ELSE
      BEGIN
        FOR J := 1 TO 36 DO
          BEGIN
            FOR I := 1 TO 80 DO
              IF EOLN(F) OR EOF(F)
              THEN A (I,J) := ' '
              ELSE READ(F,A (I,J) )
              IF NOT EOF(F)
              THEN READLN(F)
            END ;
          REPEAT
            FOR J := 1 TO 36 DO
              BEGIN
                FOR I := 1 TO 80 DO WRITE(A (I,J) );
                WRITELN
              END
            END
          END
        END
      END

```



```

REWRITE(F, 'ASCII' 'DAT') ;
WRITELN(F, ' ':6, 'ASCII CODE OF CHARACTERS') ;
Writeln(2) ;
WRITE(F, ' ':4) ;
FOR CH := '0' TO '9' DO WRITE(F, CH:3) ;
Writeln(2) ;
FOR CH := ' ' TO '~' DO
  BEGIN
    J := ORD(CH) MOD 10 ;
    I := ORD(CH) DIV 10 ;
    IF I = 0
      THEN
        BEGIN
          Writeln(2) ; WRITE(F, I:4)
        END ;
    IF CH = ' ' AND (J <> 0)
      THEN WRITE(F, I:4, ' ':J*3) ;
    WRITE(F, CH:3) ;
  END ;
Writeln(2) ;
CLOSE(F)
END

```

(* ASCII.DAT *)

ASCII CODE OF CHARACTERS									
0	1	2	3	4	5	6	7	8	9
3				"	#	\$	%	&	'
4	()	*	+	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:
6	<	=	>	⊖	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N
8	P	Q	R	S	T	U	V	W	X
9	Y	Z	[\]	^	_	◆	a
10	b	c	d	e	f	g	h	i	j
11	k	l	m	n	o	p	q	r	s
12	t	u	v	w	x	y	z	{	
								}	~

FIG FIL08 PROGRAM

这个程序 有三点值得注意:

1. ASCII 码字符中, 只有' '到 '~' 是可以打印的字符。其余的为控制字符, 不能打印或显示在终端上。而空格 (SPACE, 通常在书写上以 ' ' 或 _ 表示) 作为一个可打印的字符处理。

2. 请注意对头一个字符空格的处理方式。我们在一行中输出10个字符之后, 就要换行两次。即两行输出之间留一空白行。之后 接着打出行号。这是通过判 J 是否为零实现的, 但头一行不论对应着空格字符的 J 的值是否为零, 都得首先打出相应行号。若 J 不为零,

接着打若干空格，使空格字符正好落在相应竖列中。

3. 写一个程序，是不大容易一次成功的，特别是在稍微复杂一些的程序中，总要经过几次上机调试，才能得到正确的结果。如果我们用到行打，则在调整过程中，会打出大量无用结果，影响速度，浪费纸张。这时，最好能把结果只送到显示终端，待程序运行正确后，把最后结果在行打上输出。这时，可把我们的程序稍加改动，就是把 REWRITE(F, 'LP') 语句改为 REWRITE(F, 'TT') 其它部分不变。这时，就把要输出的结果送到终端。这在调整程序时是十分方便的。程序调好后，只要再用 'LP:' 取代 'TT:' 即可。

例8-4 许多初学程序设计的人员，不大习惯在一行上写几个语句。在他的程序中，往往一行不超过50-60个字符。行打机一般每行可打132个字符。这时，用行打输出他的程序，只在宽宽的输出纸上占用了左面一小部分地方。这时，最好能把两个程序一起打印，一个在左面，一个在右面，这就提高了纸张的利用率。下面的程序就是完成这件事情的。要注意，利用这个程序时每行长度不要超过 $132/2$ ，即66个字符。否则会遇到麻烦。程序见图 FIG FIL09 PROGRAM。

```
PROGRAM FIL09(INPUT OUTPUT,F1,F2,F3),
VAR
  I,J,K,L,M,N: INTEGER;
  CH: CHAR;
  F1,F2,F3: TEXT;
  NAME1,NAME2: ARRAY (1..6) OF CHAR;
  TYPE1,TYPE2: ARRAY (1..3) OF CHAR;
BEGIN
  WRITELN(' INPUT FIRST FILE ( NAME1 AND TYPE1 ) ( ' );
  REPEAT
    READ(NAME1,TYPE1);
    RESET(F1,NAME1,TYPE1,L);
    IF L=-1
      THEN WRITELN('NOT FOUND THIS FILE INPUT ITS NAME AGAIN')
    UNTIL L <> -1;
  WRITELN(' INPUT SECOND FILE ( NAME2 AND TYPE2 ) ( ' );
  REPEAT
    READ(NAME2,TYPE2);
    RESET(F2,NAME2,TYPE2,L);
    IF L=-1
      THEN WRITELN('NOT FOUND THIS FILE INPUT ITS NAME AGAIN')
    UNTIL L <> -1;
  REWRITE(F3,'LP:');
  WHILE (NOT EOF(F1)) AND (NOT EOF(F2)) DO
  BEGIN
    FOR I:=1 TO 65 DO
      IF NOT EOLN(F1)
        THEN
          BEGIN
```

```

        READ(F1,CH) ; WRITE(F3,CH)
    END
    ELSE WRITE(F3,' ') ;
    WHILE NOT EOLN(F2) DO
    BEGIN
        READ(F2,CH) ; WRITE(F3,CH)
    END ;
    READLN(F1) ;
    READLN(F2) ;
    Writeln(F3)
END ;
IF EOF(F2) AND NOT EOF(F1)
THEN
    WHILE NOT EOF(F1) DO
    BEGIN
        WHILE NOT EOLN(F1) DO
        BEGIN
            READ(F1,CH) ; WRITE(F3,CH)
        END ;
        READLN(F1) ;
        Writeln(F3)
    END ;
    IF EOF(F1) AND NOT EOF(F2)
    THEN
        WHILE NOT EOF(F2) DO
        BEGIN
            WRITE(F3,' ':65) ;
            WHILE NOT EOLN(F2) DO
            BEGIN
                READ(F2,CH) ; WRITE(F3,CH)
            END ;
            READLN(F2) ;
            Writeln(F3) ;
        END ;
        Writeln(F3) ;
        CLOSE(F3)
    END ;
    FIG FIL09 PROGRAM

```

对这个程序，说明三点：

1. NAME, 与 TYPE, 是文件 1 的名字和类型，所以得满足对有关标识符的规定：NAME 最多由字母开头的六个字母数字组成；TYPE, 最多由三个字母组成，通常应为计算机系统规定的文件类型（如 PAS, DAT 等），NAME, 与 TYPE, 也一样。

请注意，本程序中 在输入文件名称及其类型时，中间必须用空格分开 如 FIL01

PAS 不能输入 FIL01.PAS。

2. 把文件 1 与文件 2 逐行输出到 LP: 上。文件 1 占前 65 个字符位置, 文件 2 则从第 66 个字符位置开始。无论哪个文件的一行太长, 超出留给它的范围, 超出部分自动丢失。

3. 当一个文件已打印完毕, 而另一个尚未结束, 则只将尚未结束的文件继续输出就行了。如第二个文件先结束, 则只是简单地把第一个文件继续打印完。若第一个文件先结束, 则必须在每输出第二个文件一行前, 先输出 65 个空格, 使第二个文件的内容, 始终打印在纸张的右半部。

例8-5 用户盘上有两个文件, 它们中存放的都是整数, 并且所有的数都是按从小到大的次序排好的。请编写一个程序, 把这两个文件中的数合在一起, 按从小到大的次序排队, 把结果显示在终端上。程序见图 FIG FIL10 PROGRAM。

```
PROGRAM FIL10(INPUT, OUTPUT, F1, F2);
TYPE
  FL=FILE OF INTEGER;
VAR
  F1, F2, F3:FL;
  I1, I2, I :INTEGER;
  N1, N2, N3:ARRAY (1..6) OF CHAR;
  B1, B2 :BOOLEAN;
  PROCEDURE RF(VAR F:FL;VAR I:INTEGER;VAR B:BOOLEAN);
  BEGIN
    IF NOT B
    THEN
      BEGIN
        READ(F, I); B:=TRUE;
        IF EOLN(F)
        THEN READLN(F)
        END
      END ;
  PROCEDURE WF(VAR X:INTEGER; VAR B:BOOLEAN);
  BEGIN
    IF I MOD 6=0
    THEN WRITELN(F3, X:8)
    ELSE WRITE(F3, X:8);
    I:=I+1; B:=FALSE
  END;
BEGIN
  Writeln('PLEASE INPUT DATA FILE NAME N1 : ');
  READ(N1); RESET(F1, N1);
  Writeln('PLEASE INPUT DATA FILE NAME N2 : ');
  READ(N2); RESET(F2, N2);
  REWRITE(F3, 'TT:');
  I:=1; B1:=FALSE; B2:=FALSE;
  WHILE NOT(EOF(F1) OR EOF(F2)) DO
```

```

BEGIN
    RF(F1, I1, B1), RF(F2, I2, B2);
    IF I1 <= I2
    THEN WF(I1, B1)
    ELSE WF(I2, B2)
END;
WHILE NOT EOF(F1) DO
BEGIN
    RF(F1, I1, B1);
    IF B2 AND (I2 <= I1)
    THEN WF(I2, B2)
    ELSE WF(I1, B1)
END;
WHILE NOT EOF(F2) DO
BEGIN
    RF(F2, I2, B2);
    IF B1 AND (I1 <= I2)
    THEN WF(I1, B1)
    ELSE WF(I2, B2)
END;
IF B1
THEN WF(I1, B1);
IF B2
THEN WF(I2, B2);
CLOSE(F3); WRITELN
END.

```

```

( •          FIL10A.DAT
1  23  40  54  65  79  120  234  432  564  786  1220
4321  5432  7654  8765  9876  12345  32765 • )

```

```

( •          FIL10B.DAT
2  5  76  99  123  345  678  876  987  1000
12345  32767

```

OUTPUT:

1	2	5	23	40
54	65	76	79	99
120	123	234	345	432
564	678	789	876	987
1000	1220	4321	5432	7654
8765	9876	12345	12345	32765
32767				

FIG FIL10 PROGRAM

在这个程序中，关键是两个过程 RF 与 WF 的功能以及对两个布尔量 B1 与 B2 的使用方法。

过程 RF 有三个变量形式参数 分别为 F、I 与 B。是用来表示从哪一个文件读来一个整数，把读来的内容放在哪个变量单元。B 的作用很重要，只有当 B 为 FALSE 时，才能真正去读文件。读完后还要使 B 变为 TRUE。否则，调用 RF 就是一次空操作，不产生任何实际效果。

过程 WR 有两个变量形式参数 X 和 B（这里的 X 也可以用是数值形式参数），X 用来表示应把哪一个变量写到文件 F3 中去。每写完一次之后，都要使相应的 B 变为假。

B1、B2 的作用很重要。这可以从主程序中看清楚。当两个文件都没结束时，每次都同时去读它们。除了头一次读两个文件是真的同时都读了外（此时，B1 与 B2 均为 FALSE），之后只有一个读语句是真的去读了，另一个读语句只是一次空操作。这是因为，头一次应将两个文件的第一个成份读出来，并比较它们的值的大小，把值小的那个写到文件 F3 上，或者当读出的两个元素值相等时，就把第一个文件的元素写到文件 F3 上。请注意，写了第一个文件的元素，则要使 B1 为假，否则使 B2 为假。这样，再回过头去读时，就应该只读其头一个元素已被写入到 F3 中去的那个文件，而不能去读另外一个文件。可以在两个文件中只选择一个来读，这一点正是借助于 B1、B2 作为 RF 和 WR 的实际参数来完成的。

在有了上述说明之后，其余部分就相对容易了。值得提醒的是，在一个文件结束之后，它的最后一个元素可能尚未写入 F3，这就是为什么在程序后半部分，以及在整个程序结束之前，总在判 B1 与 B2 的值的原因。

例8-6 对 TEXT 文件入/出处理中的自动数据类型变换，并非所有人很快就能接受的。为此，我们给出一个例子，以便进一步领会一下这种变换中的问题。假定用户盘上存放着一个很长的 TEXT 文件 TXTDAT.DAT，它由几千组数据组成 每组包括 10 个实数。TXTDAT.DAT 为某一程序要处理的原始数据。如果我们要多次地使用这些数据，则首先应该把它们变换成二进制形式，并且把变换后的内容存放到另一个文件 BRYDAT.DAT 中，以后再用时，就直接使用它。而不是继续使用原来的文件。这样，就大大节省了每一次费对的数据变换。下面给出的就是完成这一变换功能的程序。

```
PROGRAM FILE11(DATAFILE, BINARYFILE, OUTPUT),
CONST
  GROUPSIZE=10,
TYPE
  INDEX=1..GROUPSIZE,
  GROUP=ARRAY (INDEX) OF REAL,
VAR
  DATAFILE:TEXT,
  BINARYFILE:FILE OF GROUP,
  IX:INDEX,
  GROUPCOUNT:INTEGER,
BEGIN
  RESET(DATAFILE, 'TXTDAT', 'DAT'),
  REWRITE(BINARYFILE, 'BRYDAT', 'DAT'),
  GROUPCOUNT:=1,
  WHILE NOT EOF(DATAFILE)DO
    BEGIN
```

```

IX:=0,
REPEAT
  IX:=IX+1,
  READ(DATAFILE, BINARYFILE' [IX] )
  UNTIL(IX=GROUPSIZE) OR EOF(DATAFILE),
IF EOF(DATAFILE)
  THEN WRITELN('FILE ENDS WITH SHORT GROUP')
  ELSE GROUPCOUNT:=GROUPCOUNT+1,
  PUT(BINARYFILE)
END,
WRITELN(GROUPCOUNT,4, 'GROUPS CONVERTED'),
WRITELN('LAST GROUP CONTAINS', IX, 3, ' REAL NUMBER(S)'),
CLOSE(BINARYFILE)
END

```

```

(*   TXTDAT.DAT
  1  2  3  4  5  6  7  8  9 10
12 23 45 34 56 78 98 90 45678
1234567 890
*)

```

OUTPUT:

```

FILE ENDS WITH SHORT GROUP
 3 GROUPS CONVERTED
LAST GROUP CONTAINS 1 REAL NUMBER(S)

```

```

(*   BRYDAT.DAT
000/040200 000000 040400 000000 040500 000000 ...
...
120/042536 100000

```

FIG FIL11 PROGRAM

这个程序中，对输入输出文件均未考虑有行结束问题，这是一种特殊情况，使处理程序变得简单些。

这里顺便提出两个问题，一个就是在使用 BRYDAT.DAT 文件时，才能再简单地用 EOF (BINARYFILE) 来判文件结束。前面曾说过 EOF (F) 的办法只适用于 TEXT 文件。为此，可以使用已经在终端上显示出来的 GROUPCOUNT 与 IX 的值，通过计算读出文件的元素个数的办法，来了解文件是否应该结束。二是应该用 GET (BINARYFILE) 到这个文件中取来一个实数，而不能用 READ，务请注意。

本章习题

1. 在磁盘中以文件形式建立一个正弦函数表，其格式如下，

```

              THE LIST OF SIN(X)
1 SIN(I)   1 SIN(I)   1 SIN(I)   1 SIN(I)   1 SIN(I)
0 0.0000   1 0.0175   2 0.0349   3 0.0523   4 0.0698
5 0.0872   6 0.1045   7 0.1219   8 0.1392   9 0.1564

```

.....

到359°为止。

2. 从磁盘中调任意一个文本文件,搜索字符串'WRITE ('和'WRITELN (',分别将它们改为'WRITE (M,'和'WRITELN (M,',然后作为一个新的文件,用原名称存入磁盘中。

3. 从磁盘中调任意一个文本文件,将其两个'#'之间的全部文件内容在行式打印机上打印输出。

4. 从磁盘中调任意一个文件,将其从字符串1 (不超过3个字符)到字符串2 (不超过3个字符)的全部文件内容 (包括字符串1和2)作为一个新文件存入磁盘中。

5. 在磁盘中已有50个学生情况的文件,包括学号、姓名、性别、年龄和五门课程的成绩。编一程序,以建立以下四个文件:

(1) 女生情况的文件;

(2) 按年龄从小到大排列的学生情况的文件;

(3) 按五门课程的平均成绩的高低排列的学生情况 (加五门平均成绩一栏) 的文件;

(4) 按年龄 (比如17..25), 按各门课程及五门平均成绩的分数段 (60分以下, 60—70, 71—80, 81—90, 90分以上) 进行人数统计的文件。(可参考第十一章第十个程序)

第九章 指 针

在前面几章，我们已经学了基本类型与构造类型的各种数据，它们都属于静态的数据。PASCAL 语言中的静态数据有两个重要特性：

第一，程序员在程序的说明部分，必须确定每个数据的标识符与类型，并在程序的执行部分中，通过定义好的标识符对它们进行访问；

第二，这些数据都有自己的生命期限，在它们的整个生命期限之内，数据个数保持不变。

在这一章，我们将要学习另外一种类型的数据——动态数据。它们与静态数据正好相反。首先，程序员无需在程序的说明部分给出这类数据的标识符，因此，也就不能通过变量名来访问这一类数据；其次，这类数据是在程序的运行过程中动态地建立起来的，也可以动态地撤消其全部或其中的一部分。随着程序的运行，它们可以不断地改变自己的数据个数，我们习惯上称这样一类数据为动态数据，称这类变量为动态变量。

在本章，我们将重点介绍如下三项内容：为什么要用动态变量，怎样在 PASCAL 语言的程序中定义动态变量；如何正确使用动态变量。最后再给出几个例证程序。

§1. 从静态变量到动态变量

在用计算机处理的许多问题中，经常用到一些表格数据，如链表结构数据。当选用了 PASCAL 语言作为程序设计工具之后，应如何表示这种链表数据结构呢？回忆一下我们前面学习过的几种数据类型，很容易想到，可能有两种解决办法：

(一) 用数组表示，如：

VAR

LIST: ARRAY [1..LISTSIZE] OF OBJECT;

(二) 或者用文件表示，如：

VAR

LIST: FILE OF OBJECT;

这里的 OBJECT 表示的是变量 LIST 的每一个元素的类型，LISTSIZE 表示的组成 LIST 数据的元素个数。

在某些情况下，采用这两种办法会给我们带来许多问题。

我们先看第一种解决办法，即用数组实现链表结构数据时将遇到的问题。

我们在学习数组类型时已说过，数组必须在程序的说明部分加以说明，确定数组元素类型，下标类型及下标的上下界。这就是说，必须在程序的说明部分中就确定好 LISTSIZ 的值。但是，在解决某些问题的有关程序中，不一定能在说明部分就给出这个数值。例如，我们写一个 PASCAL 语言的编译程序，必须处理好用户程序中的说明部分说明过的常量，类型，变量，过程，要登记保存好它的名称、类型、数值、地址或参数等等。我们的编译程序应满足每个用户的要求，而每个用户的程序所需要的常量、变量、过程和函数等的数目和

类型有极大差别，少的只有一两个，多的可达数百个。我们显然设法用数组方式来最好地适应这种情况。如果在我们的编译程序中，有关数组的元素个数留少了，则在处理说明了很多常量、类型、变量等的程序时，就势必会遇到数组越界的问题。为解决这个矛盾，可以为每个有关数组都留下“足够”的存贮空间。那么对于只说明了少量的常量、类型、变量等的程序来说，编译程序又浪费了大量的存贮单元。这是一对难以在数组概念内解决的矛盾。比较理想的解决方案是设法避开在编译程序内部事先留出若干存贮单元的做法，只是根据出现在用户程序的说明部分中的常量、变量、过程和函数等的实际个数来动态地登记它们，遇到一个登记一个。这就要求我们有办法在编译程序的执行过程中动态地建立一些数据。

应用数组的另一个困难是，它很难实现在一组按序排列的数据中的适当位置插进一个或几个数据，或者删除它们中的某一个或几个。例如，用 PASCAL 语言设计某台机器的操作系统的设备管理程序时，我们可以把说明每台设备的设备名（或设备号），设备的特性，某些有关程序的起始地址，该设备所使用的缓冲区地址、缓冲区大小、缓冲区中所存信息个数等有关信息定义为一个记录，通常称它为设备控制表，简记为 DCT (DEVICE CONTROL TABLE)。假定每台设备都有自己的一个 DCT 表，如果我们的系统能支持几台不同的设备，为了管理这些设备，我们就要把这几个 DCT 表组织起来。假定我们在每台设备发生中断时，按处理这些中断的先后次序，把它们排列起来，则似乎可以如下说明：

```
TYPE DCT=RECORD
```

```
    NAME; ALFA; (设备名)
```

```
    CHARACTERS; ARRAY [1..16] OF BOOLEAN; (设备特性)
```

```
    INTERRUPT; 0..177777; (中断处理程序入口)
```

```
    BUFFERADS; 0..177777; (缓冲区起始地址)
```

```
    BUFFERSIZE; 1..180; (缓冲区大小)
```

```
    NUMBER      ; 0..180; (缓冲区现在的信息个数)
```

```
    .....
```

```
END;
```

```
VAR DCTLIST; ARRAY [1..n] OF DCT;
```

粗看起来，这种解决似乎没有问题。但是，它有着很大的不足之处。作为操作系统，它应能管理它所支持的全部 n 台设备，则数组 DCTLIST 就应该由 n 个元素组成。但实际情况是，很难找到在哪个用户买的一个计算机系统中，配备了这全部的 n 台外设。通常情形是，每个用户只配备他们所需要的若干台。这时就需要删去本系统中未配置的那些设备的 DCT 表，就要在 DCTLIST 中造成许多“空洞”。这不仅需要特殊加以标志，而且白白地浪费许多存贮单元。这是第一个问题。第二个问题是，有些用户还需要增加原来系统不支持的，自己专配的特殊设备。操作系统应给出某种手段，使用户可以把新的设备的 DCT 表加到 DCTLIST 中去，这也是个很难实现的事情。有没有地方（即多余的存贮单元）可以加入，如何方便地加入等，都存在某些问题。

那么，采用文件结构又如何呢？也有问题。文件对于解决数据元素的个数不断增加或减少的问题，是能够做得到的。但是，它要占外存，读写速度慢，尤其是要在文件当中的不同位置加入或删除一个或几个数据元素，是非常不方便的，需要修改，重写整个文件。

总而言之，要解决类似上面两个例子提出的问题，采用静态的数据结构是很困难的。为此，在 PASCAL 和其它某些语言中，还给出了一个新的数据的类型——指针类型。通过指针类型变量，就可以在程序的执行过程中，按照用户的要求动态地建立一些变量。变量的个数

已不再是一种限制。它能方便地高效率地加入或删除若干数据，又避免了加入或删除过程中，对存储单元进行管理的许多细节问题，以及前面提到的存储单元的浪费问题。

在具体介绍指针概念之前，先看一个简单的程序。

```
PROGRAM PNT01A(INPUT, OUTPUT),
TYPE
  LINK=^DATA,
  DATA=RECORD
    NEXT:LINK,
    INT:INTEGER
  END,
VAR
  P, Q:LINK,
  I:INTEGER,
BEGIN
  (* PART 1 INPUT SOME INTEGERS
  AND MEMORIZE THEM IN DYNAMIC DATA *)
  Q := NIL,
  REPEAT
    READ(I),
    NEW(P),
    WITH P^ DO
      BEGIN
        NEXT:=Q, INT:=I
      END,
    Q := P
  UNTIL I=-1,
  (* PART 2 OUTPUT ALL INPUTTED INTEGERS *)
  WHILE P<>NIL DO
    BEGIN
      Writeln(P^ INT),
      P:=P^.NEXT
    END
  END.
INPUT:
32767 2345 5078 166 -15 -1
OUTPUT:
-1
-15
166
5078
2345
32767
FIG PNT01A PROGRAM
```

在这个程序的类型说明中，定义了两种数据类型，即指针类型 LINK 和记录类型 DATA：在变量说明部分，定义了两个 LINK 类型的指针变量 P 和 Q，又定义了一个整型变量 I，

这个程序的功能是，它可以连续地读入一序列的整数，并以动态变量的形式把它们记录下来。当读入的整数为 -1 时，停止输入；然后再把输入的全部整数按与原来输入完全相反的次序输出来。这个程序中用到的新的符号与概念，将在后面部分作详细介绍。在开始读后面的内容之前，先仔细读一读与思考一下这个程序是有益的，而在读后面内容的过程中，再回过头来反复地看一下这个程序，会有助于对某些概念的理解。

§2. 指 针

指针 (POINTER)，又称指示器，是一种数据类型。它属于简单类型。但 POINTER 这个单词本身不是 PASCAL 语言的保留关键字，它与 INTEGER, REAL, 以及 BOOLEAN 等词不同，后者是 PASCAL 的保留关键字，可以直接出现在 PASCAL 程序的说明部分。如：

```
VAR
    I, J, K, INTEGER;
    B, BOOLEAN;
```

如果你想定义 P 为指针类型变量，则不能采用如下办法：

```
VAR
    P, POINTER;
```

因为机器不认识 POINTER 这个词。不把 POINTER 作为一个关键字是有原因的。指针，顾名思义，是说它指向一个数据，那么，指向什么样的数据呢？我们无法用 POINTER 这个词本身来表示。为此，可以在程序的类型说明部分，首先定义一个具体的指针类型：

```
TYPE
    LINK = ↑ OBJECT;
```

我们可以把上一说明读作为：定义 LINK 为指向一个 OBJECT 类型数据的指针类型。在进行这项说明时，我们在“=”与“OBJECT”之间用了一个向上指的箭头“↑”。（在 PDP-11/03 机上用“^”代替），也正是借助于这个“↑”字符，才表明这里定义的是一个指针类型。等号左边的 LINK 是一个指针类型标识符，等号右边的 OBJECT 是一个数据类型的标识符。“↑”表示 LINK 与一个 OBJECT 类型数据之间的关系。

下面就涉及到有关 OBJECT 本身的讨论。

前面已经提到，OBJECT 是类型标志符。我们学过的标准类型有 INTEGER, REAL, CHAR 和 BOOLEAN。我们也可以用它们来定义指针类型。先看下面一个简单的例子。

```
PROGRAM PNT02(OUTPUT);
TYPE
    LINK = ^INTEGER;
    LK = ^REAL;
VAR
    I, J: INTEGER;
```

```

R:REAL,
L:LINK,
LK1:LK,
BEGIN
  I:=-1,
  J:=10,
  R:=0.52,
  L:=@I;    (*TRANSFER THE ADDRESS OF I INTO L*)
  WRITELN(I, L↑),
  J:=L↑,
  WRITELN(J, L↑),
  LK1:=@R;  (*TRANSFER THE ADDRESS OF R INTO LK1*)
  WRITELN(R, LK1↑),
END.
OUTPUT:

```

```

      -1      -1
5.200000E-01 5.200000E-01
      -1      -1

```

FIG PNT02 PROGRAM

在这个程序的类型说明部分，定义了指针类型标识符 LINK 和 LK，它们分别对应于整数和实数。在变量说明部分，定义 L 为 LINK 类型的指针，它可以指向整型数据。又定义 LK₁ 为 LK 类型的指针，它可以指向实型数据。在程序的执行部分，通过 L:=@I 这一赋值语句将整型变量 I 的地址传送到 L 中，这时 L 就指向 I，也就是说 L↑代表的就是 I 变量。这可以从下一个输出语句中看到，输出的 I 与 L↑都是 -1，这是对的。我们还可以用 J:=L↑完成把 I 的值赋给 J，对于 LK₁ 与 R 的关系也一样。要指出的是，这样做有什么必要呢？从 PASCAL 语言程序设计一级来看，确实不会给我们带来任何好处。但可以用这种办法，把 PASCAL 程序中的一些变量的地址，传送给系统程序中的 MACRO 语言写的程序段。此种办法，也可以类似地用于所有构造型数据。

更详细的内容将在本章第五节进行说明。

从建立动态数据的要求出发，选择 OBJECT 为标准类型或前面讲过的构造类型是不行的。因为那种办法只能实现变量的地址传送，达不到建立一“串”动态数据的目的。为了建立动态数据，就必须使 OBJECT 类型中包括其类型为相应指针类型 LINK 的数据成分。如果 OBJECT 只包含一个 LINK 类型数据，这种数据结构是没有任何用处的，如果 OBJECT 只包含几个 LINK 类型数据成分（即 OBJECT 是由 LINK 类型数据组成的数组），这种数据结构也无多大用处。就是说，OBJECT 中还应该包括有 LINK 类型之外的其它类型的数据，在 PASCAL 语言中能满足这种要求的唯一数据结构，就是记录了，所以 OBJECT 通常是 RECORD 类型。例如：

```

TYPE
  LINK = 'OBJECT,
  OBJECT = RECORD
      NEXT: LINK;

```

DATA; DATATYPE

END;

在 PASCAL 语言中, 在用到任何标识符之前, 一般都必须先对这一标识符加以说明, 但对于以下两种情况:

——当存在着向前调用的过程时;

——当定义指针类型时, 则允许在说明某个标识符之前就用到它。这里我们就具体地看到了这后一种情况。要定义 LINK, 就要用到 OBJECT, 要说明 OBJECT, 又要用到 LINK, 因此没有办法满足在使用一个标识符之前, 必须对它首先加以说明的一般规定。为此 PASCAL 语言的设计者就制定一个新的规则, 就是首先定义指针类型, 然后再说明这个指针类型对应的记录类型。这样我们就可以在说明 OBJECT 之前, 就运用 OBJECT 去定义 LINK。

在定义好类型之后, 就可以用如下方式说明变量:

VAR

P, Q, R; LINK;

在这里, P, Q, R 是三个指向 OBJECT 类型数据的指针类型变量。指针类型变量简称指针。有了指针, 就可以借助于它们来动态地建立 OBJECT 类型数据。在介绍怎么用它们建立 OBJECT 类型的动态变量之前, 我们明确几个概念, 熟悉几种符号。

(1) P, Q, R 是指向一个 OBJECT 类型数据的三个指针, 图 9.2-1 示意图可以表示这种关系:

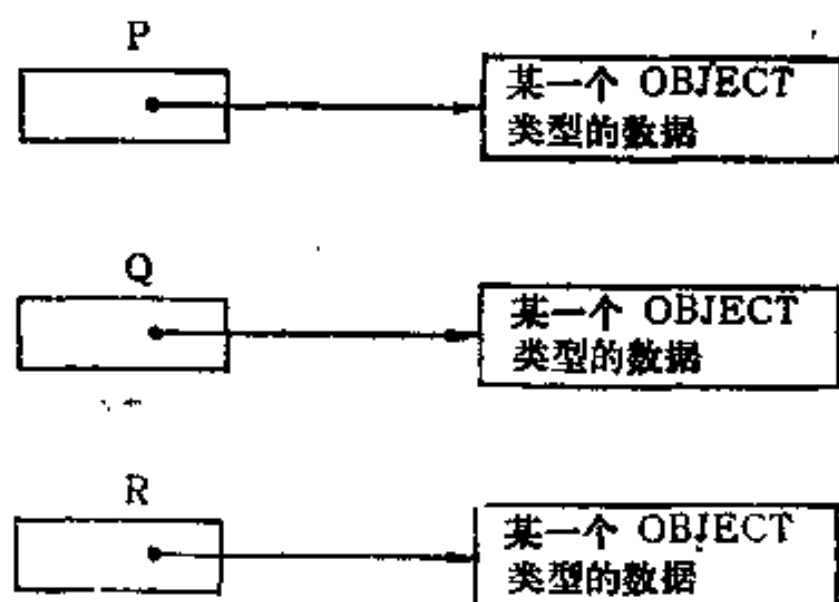


图 9.2-1 指针和它指向的对象

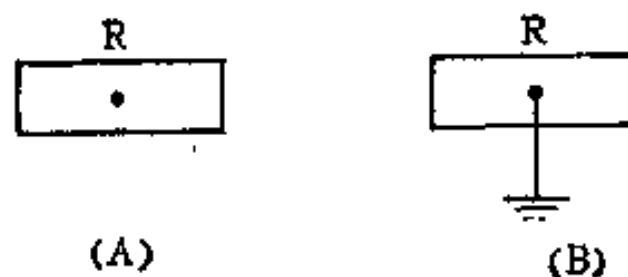


图 9.2-2 空指针的表示

如果我们要表示它们中的一个, 如 R, 并未指向任何一个 OBJECT 类型的数据, 则可以用如下一个赋值语句:

R := NIL

具有值 NIL 的指示器, 用图 9.2-2 表示。

在某些书中, 也有用类似于电子线路中的地线符号表示的, 如图 9.2-2(B) 所示。

在数据结构的理论讨论中, 更多的却是用符号 ^ 来表示这种情况。我们常常称未指向任何数据的指针变量为空指针, 或者说它的内容是空的。

(2) 怎样才能使 P, Q, R 指向一个 OBJECT 类型的数据呢? 又怎样才能访问一个 OBJECT 类型的数据本身呢? 这是人们想要解决的疑难问题。大家可能已经注意到, 在这一章里, 总是提 OBJECT 类型的数据, 而不是提数据 A, 数据 B 等等。这是因为我们无法

给 OBJECT 类型的每一个数据命名。换言之，OBJECT 类型的数据本身没有名字，因此也就无法通过名字去访问它们。但是，在定义了指针类型后，可以通过指向它们的指针类型变量 P, Q, R 等引用它们。因为 P 指向一个 OBJECT 类型的变量，说的就是存放一个 OBJECT 类型变量所用的若干个存储单元的首址放在 P 里。因此，要访问这个 OBJECT 型变量，只能通过 P。这时，我们就用 $P \uparrow$ 表示 P 当前指向的那个变量本身。这里我们要严格区分 P 和 $P \uparrow$ 的意义，P 是指针变量，它里面存放的是一个地址， $P \uparrow$ 是 P 所指向的一个数据。请看下图给出的一个例子，看一下 $P := Q$ 和 $P \uparrow := Q \uparrow$ 的不同作用。我们假定 OBJECT 中的数据域中的 DATATYPE 为 ALFA，则可以有如图 9.2-3 所示的情况。

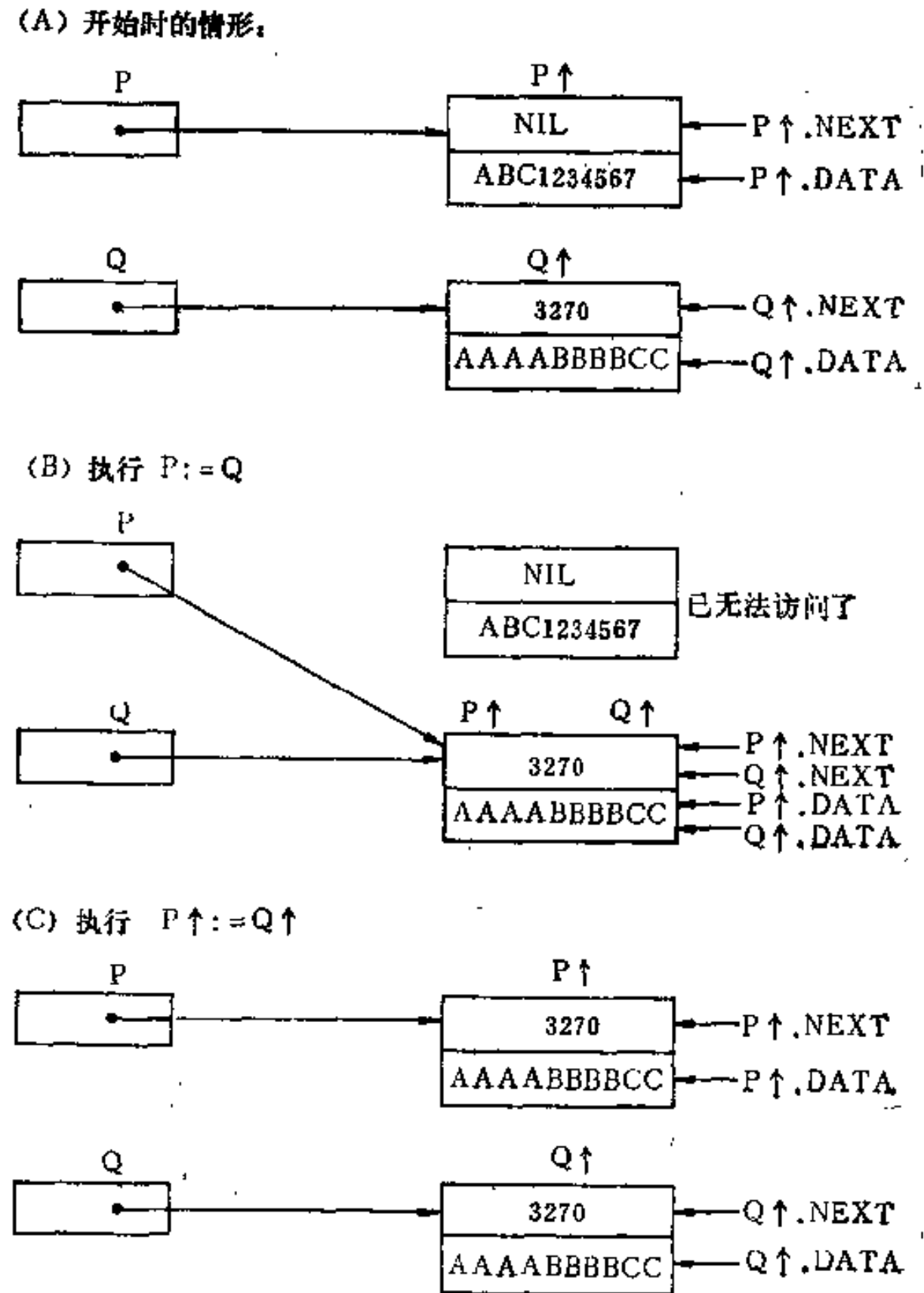


图 9.2-3 指针及其赋值情形

从图上不难发现，可以向指针变量本身赋值 ($P := Q$ 的情形)，也可以向通过指针所引用的动态变量赋值 ($P \uparrow := Q \uparrow$ 的情形)。但应注意，只准在同一类型的指针变量间彼此赋值，只准在同一类型的动态变量之间彼此赋值。这一点在后面部分还要详细讨论。

§3 两个标准过程NEW(P)和DISPOSE(P)

为了通过指针对动态数据进行管理，在PASCAL语言中给出了两个标准过程NEW和DISPOSE。它们的参数(Argument)是指针类型变量(以下简称为指针)。如NEW(P)，DISPOSE(P)等都是正确的调用方式。

NEW(P)的作用，是要建立一个经指针P所引用的动态变量。这一过程结束以后，就完成了如下两件事情：第一，从内存空闲区(Free Memory)找出用于存放一个对应动态变量的若干存贮单元；第二，把这若干存贮单元的头一个(字的)地址写入指针P中。在这之后，就可以通过P来引用新建的这个变量。有三点注意事项，必须在这里强调说明：

1. 此时P'的内容尚未确定，因此应把要写进去的内容，以如下赋值方式写进去：

$P' \cdot \text{NEXT} := \text{NIL};$

$P' \cdot \text{DATA} := \text{'ABC1234567'};$

2. 如果后面又要通过P建立下一个动态数据，一定要在新的NEW(P)调用之前，把P本身的值(刚建立的动态数据的起始地址)保存好，这可以通过 $Q := P$ 来完成，否则，完成新的NEW(P)调用之后，P中存放的就是新建数据的起始地址，原来的内容丢失了。也就是说，已没有任何办法再对上一次建立的那个数据进行访问了。

3. 一个指针每次只能指向一个相应的数据，而我们的目的，却是用少数几个(至少要两个)同类指针建立与引用许多个同类动态数据。这似乎有矛盾。回忆一下在§2开始部分讨论应如何定义指针类型时，曾说到：“从建立动态数据的要求出发，OBJECT类型中应包括其类型为相应指针类型LINK的数据成分”。这个数据成分本身就是一个指针。它可以指向它前面一个动态数据。如果我们依照一串动态数据生成的次序讨论，我们就可以说，第*i*次生成的数据中的指针，指向第*i*-1次生成的数据，而第*i*+1次生成的数据中的指针又指向第*i*次生成的数据。这就是说，被生成的同类动态数据彼此是勾连在一起的。这可以简明表示如图9.3-1。

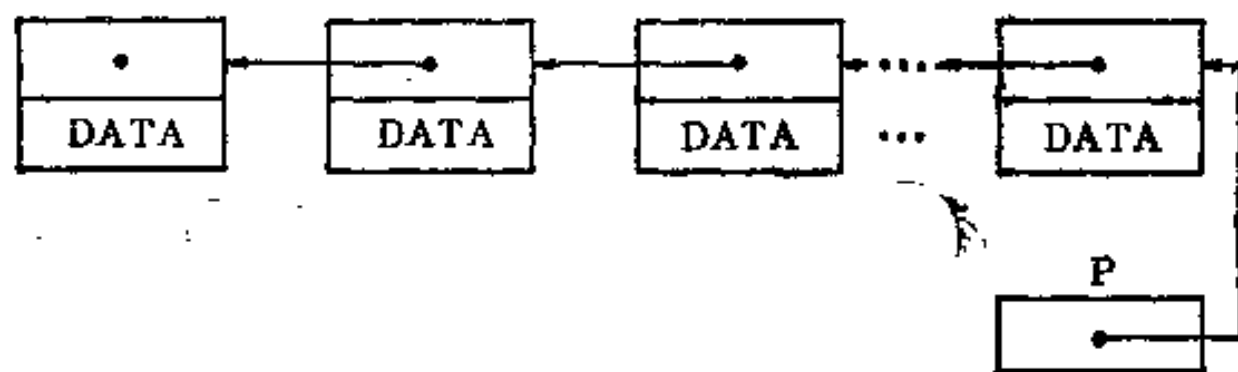


图 9.3-1 链表结构的动态数据

这时只要有办法标明这一串数据的起始与终止位置就行了，如果它们都是通过调用NEW(P)生成的，则P一定指向最后生成的那个数据。而头一次生成的那个数据的NEXT域不再指向其它数据，用NIL值向它赋值就行了。

回忆一下上面提到的注意事项2，就会懂得，在完成第一次NEW(P)调用与向P'写入相应内容之后，应立即完成 $Q := P$ 操作，之后才可以进行第二次NEW(P)调用，并接着完成 $P' \cdot \text{NEXT} := Q$ 操作，这时刚生成的数据中的指针(就是 $P' \cdot \text{NEXT}$)就指向上一次生成的数据。如果还要为 $P' \cdot \text{DATA}$ 赋值，就完成之。但应注意的是，不应忘记再次执行 $Q := P$ 操作。这样，建立同类型的一串动态数据的操作就可以用循环方式来完成。

DISPOSE(P)是NEW(P)的逆过程，它的作用是取消P当前所指向的那个数据。由于这个过程未给应用者带来更多的好处，所以在标准PASCAL语言中已经取消了。然而，在某些机器上运行的PASCAL语言中，仍保留着它。在应用DISPOSE(P)前，首先应使P指向要取消的那个数据，这样才能保证取消的正是你要取消的那个数据。然而这还不够，还要完成一件事，就是还应保证，在取消了其中的一个数据之后，不能使原来勾链在一起的同类数据被断为两截。请看如下一个最简单的例子：

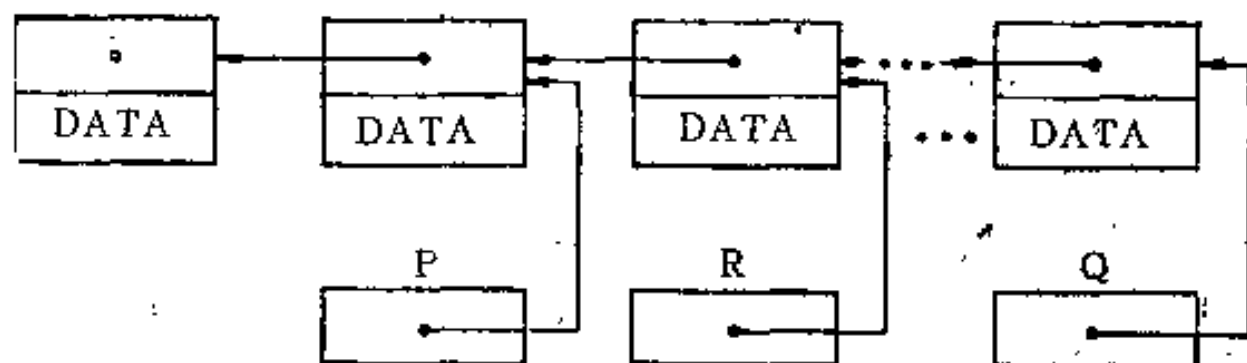


图 9.3-2 链表中某个数据的取消

如果我们要取消第二个数据，首先找到第二个数据位置，假如我们让P指向它，如上图所示。但应记住，在取消这个数据之后，应使第一个数据和第三个数据勾链起来，否则，一个链表就被截断为两截了。为此应把第一个数据的地址(现在正保存在第二个数据的NEXT域中)，写到第三个数据的NEXT域中。但若P已指向第二个数据，再要找第三个数据则不那么容易了。所以从效率方面考虑，还要用另一个指示器R指向第三个数据。如上图所示。这时就可以执行如下操作：

```
R^.NEXT:=P^.NEXT;
DISPOSE(P);
```

当然也可以用 $R^.NEXT:=R^.NEXT^.NEXT$ 办法实现，这两种办法之间是有一些差别的。

两种方法都删除了第二个数据，并保证剩下来的那些数据仍正确的勾链在一起。

请读者注意，在我们上面选用的例子中，是后建立的数据中的指针指向先建立的一个数据，当然也完全可以让先建立的数据中的指针指向紧接着建立的下一个数据，或者在一个数据中，建立两个指针，一个指向它前面的那个数据，另一个指向它后面的那个数据。在数据结构中，通常把用单个指针勾链起来的数据称为单向链表，把用两个指针勾链起来的数据称为双向链表。在程序中应选用哪种形式，往往取决于实际需要与程序员的习惯。

§4 指针应用的几个简单程序

指针是一种很重要的数据类型，对于它的应用举例如下。

例9-1 先回过头去看一下在本章§2开始部分给出的那个程序和那里的一点说明。

```
PROGRAM PNT01A(INPUT, OUTPUT);
TYPE
  LINK='DATA';
  DATA=RECORD
    NEXT:LINK;
    INT:INTEGER
```



```

        END;
VAR
    P, Q: LINK;
    I: INTEGER;
BEGIN
    (• PART 1 .....INPUT SOME INTEGERS
      AND MEMORIZE THEM IN DYNAMIC DATA •)
    Q:=NIL;
    REPEAT
        READ(I);
        NEW(P);
        WITH P↑ DO
            BEGIN
                NEXT:=Q;  INT:=I
            END;
        Q:=P
    UNTIL I=-1;
    (• PART 2 .....OUTPUT ALL INPUTTED INTEGERS •)
    WHILE P<>NIL DO
        BEGIN
            WRITELN(P↑.INT);  P:=P↑.NEXT
        END
    END.
INPUT:
32767 2345 5078 166 -15 -1
OUTPUT:

```

```

        -1
        -15
        166
        5078
        2345
        32767

```

FIG PNT01A PROGRAM

这里再说明一下程序的执行过程。为了建立一“串”动态数据，用了 REPEAT 语句。在 READ(I) 语句之后，是 NEW(P) 语句。这是调用标准过程 NEW，P 是它的实际参数。它的作用是建立一个动态数据，即先到内存空闲区分配两个存贮单元（一个字作为 NEXT 域，另一个字作为 INT 域），并把第一个字的地址写入 P 中。下面的开域语句的功能是，把读来的整数 I 的值写入 INT 域，这是为新建立的这个动态变量赋值。此时，NEXT 域尚没有确定的值，我们把 Q 的值赋给它。第一次时，Q 为 NIL，以后 Q 中存放的是前一个动态数据的地址。下面的 Q:=P 是把 P 的当前值（新建立的动态数据的起始地址），保存到 Q 中，以便在执行下一个 NEW(P) 时，可以把新的 P 的值写入上一个动态数据的 NEXT 域中 即 Q↑.NEXT 中，以完成动态数据的勾链操作。下面判读来的这个整数值是否为 -1，以决定是否结束 REPEAT 语句。所以，该程序的 PART 1 建立的动态数据的形式是：

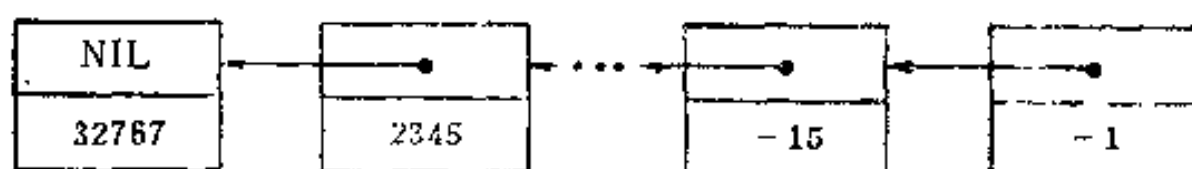


图 4-4-1 动态数据的一种形式

程序的 PART 2 部分容易懂。从最后一个动态数据开始输出，并依据它的 NEXT 域，逐步往前找，直到全部输出完为止。

从输出结果可以看到，这是按与输入相反的次序输出结果的。当然也可以按原来输入的次序输出结果，这时，要在建立动态数据的过程中，使这次建立的数据的 NEXT 域指向下一次建立的数据。这可以用下面的一个程序实现。

```

PROGRAM PNT01B(INPUT, OUTPUT),
TYPE
  LINK=^DATA,
  DATA=RECORD
    NEXT:LINK,
    INT:INTEGER
  END,
VAR
  P, Q, R:LINK,
  I:INTEGER,
BEGIN
  (• PART 1 .....INPUT SOME INTEGERS
    AND MEMORIZE THEM IN DYNAMIC DATA •)
  READ(I),
  NEW(P);  R:=P,  Q:=P,
  P^INT:=I,
  REPEAT
    READ(I),
    NEW(P),
    P^INT:=I
    Q^NEXT:=P,
    Q:=P,
  UNTIL I=-1,
  P^NEXT:=NIL,
  P^INT:=-1,
  (• PART 2 .....OUTPUT ALL INPUTTED INTEGERS •)
  P:=R,
  WHILE P<>NIL DO
    BEGIN
      WRITELN(P^INT)
      P:=P^NEXT
    END
  END.

```

INPUT:

32767 2345 5078 166 -15 -1

OUTPUT:

32767

2345

5078

166

-15

-1

FIG PNT01B PROGRAM

这里再说明一下程序的执行过程。在 READ(I) 语句之后，是 NEW(P) 语句。这是调用标准过程 NEW，P 是它的实际参数。它的作用是建立一个动态数据，即先到内存空闲区分配两个存储单元（一个字作为 NEXT 域，一个字作为 INT 域用），并把第一个字的地址存入 P 中。R:=P 是记下第一个动态数据的地址。下面就进入重复语句，其具体功能是首先把读入的整数值写入 P⁺.INT 域。这是为新建立的一个动态变量赋值。此时，动态数据的 NEXT 域尚没有确定的值。Q:=P，这是把 P 的当前值（新建立的动态数据的起始地址）保存到 Q 中，以便在执行下一个 NEW(P) 时，可以把新的 P 的值写入上一个动态数据的 NEXT 域中，即 Q⁺.NEXT 中，以完成动态数据的勾链操作。下面判读来的这个整数，若其值为 -1，重复结束。其后的两个语句 P⁺.NEXT:=NIL；P⁺.INT:=-1 是把最后读来的一个整数写入 P⁺.INT 域，并把 P⁺.NEXT 赋值为 NIL。所以，该程序的 PART 1 建立的动态数据的形式是：

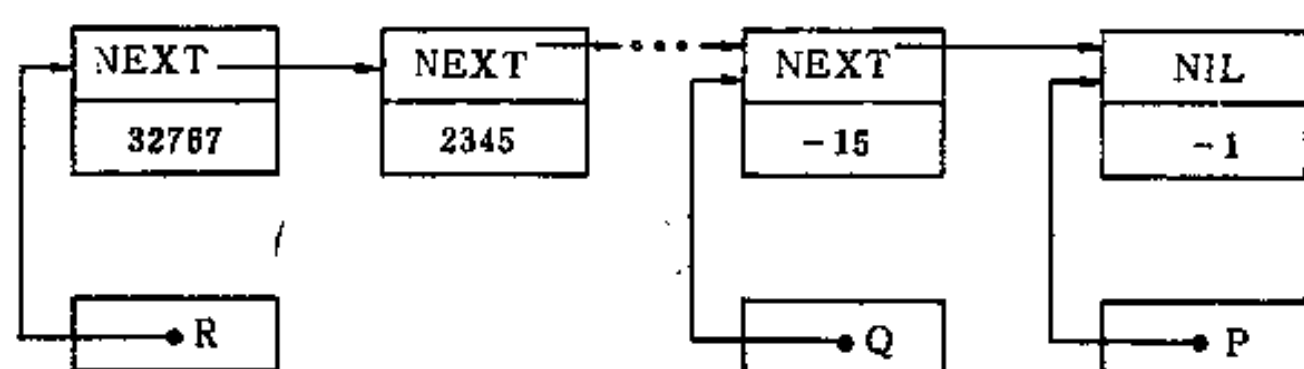


图 9.4-2 动态数据的另一种形式

从图上可以看到，R 指向头一个动态数据，P 总是指向最后的一个动态数据，而 Q 则指向倒数第二个数据。

程序的 PART 2 容易懂，首先把 P 移到第一个数据那里 (P:=R 语句完成)，然后输出它的 INT 域的值，并使 P 指向下一个数据 (P:=P⁺.NEXT 语句完成)，直到全部输出完毕为止。

从这个简单程序可以看到，用指针来建立动态变量是方便的。动态变量的数目可以变化。但是，它同时却带来另一个问题，就是内存的有效使用率降低了。NEXT 域只是为了把这些动态数据勾链起来才使用的。因此，对某些数据到底应该应用动态变量还是静态变量表示，要依具体情况而定。

看下面一个例子。在处理某些问题的程序中，会产生并反复运用到大量的中间结果，我们常常以数组形式来存放它们。如：用计算机进行逻辑化简的程序就是如此。在这里就遇到一个不好解决的问题，就是这些中间结果的个数随着输入的原始数据个数（如变量个数）的增加而迅速增加，而且其准确数在运行前也难以具体知道。这就给说明数组的上下界带来困

难。说明的少了，运行中会遇到数组越界错误，说明得多了，可能要浪费大量内存空间。如果把静态数组与动态数据结合起来用，就可以解决这类矛盾。看下面一个示意性的程序。

```

PROGRAM PNT03(INPUT, OUTPUT),
TYPE
  AR=ARRAY (0..255) OF INTEGER,
  LINK=↑AB,
  AB=RECORD
    NEXT:LINK,
    DATA:AR
  END,
VAR
  P, R:LINK,
  ARY:AR,
  I:4 12,
  A B Y, INDEX:INTEGER,
  PROCEDURE ENTER(VAR Q:LINK,N:INTEGER),
  BEGIN
    N:=N-1,
    IF N>-1
    THEN
      BEGIN
        NEW(Q),  ENTER(Q↑.NEXT, N)
      END
    ELSE Q:=NIL
  END,
BEGIN
  (* PART 1, CREATE THE DYNAMIC DATA ARRAY IF NECESSARY *)
  READ(I),
  B:=1,
  FOR A:=1 TO I DO B:=B*2,
  IF I>8
  THEN ENTER(P, B DIV 256-1),
  (* PART 2, INITIALIZE EACH ARRAY *)
  FOR INDEX:=0 TO B-1 DO
  IF INDEX<256
  THEN ARY (INDEX) :=INDEX
  ELSE
    BEGIN
      IF INDEX=256
      THEN R:=P
      ELSE
        IF INDEX MOD 256=0
        THEN R:=R↑.NEXT,
        A:=INDEX MOD 256,

```

```

        R+.DATA (A) :=INDEX,
    END,
    (* PART 3, OUTPUT THE VALUE OF ALL ARRAY'S ELEMENTS *)
FOR INDEX:=0 TO B-1 DO
    BEGIN
        IF INDEX MOD 10=0
            THEN WRITELN,
        IF INDEX<256
            THEN WRITE(ARY (INDEX) :5)
            ELSE
                BEGIN
                    IF INDEX=256
                        THEN R:=P
                    ELSE
                        IF INDEX MOD 256=0
                            THEN R:=R+.NEXT,
                        A:=INDEX MOD 256,
                        WRITE(R+.DATA (A) :5)
                    END
                END,
        WRITELN
    END.
INPUT:
    4
OUTPUT:
    0   1   2   3   4   5   6   7   8   9
   10  11  12  13  14  15
INPUT:
    12
OUTPUT:
    0   1   2   3   4   5   6   7   8   9
   10  11  12  13  14  15  16  17  18  19
    ....
   4080 4081 4082 4083 4084 4085 4086 4087 4088 4089
   4090 4091 4092 4093 4094 4095
    FIG PNT03  PROGRAM

```

这个程序的功能是，读进一个 I（其值在4~12之间），然后按照 I 的大小决定用多大的数组，数组的元素个数应等于 2^I ，就是说应在16—4096之间。怎么选定这个数组的上下界呢？在程序中 我们先定义了一个静态数组，其下标范围为0~255，就是说，当 I 在4—8之间变化时，用它就够了。当然 I 为 4 时，要浪费204个存储单元，还可以“忍受”。当 $I > 8$ 时，则动态地建立数组数据，I 每增加 1，就要建一个或几个 256 个元素的动态数组数据。这是在程序的 PART 1 部分完成的。在以后应用它们时，当然不如应用静态数组那么方便，先按下标找到对应的动态数据，然后才能访问那个相应单元。说的具体些就是，如果所用的下

标 INDEX 小于 256, 只用静态数组就够了, 如 `WRITE (ARY(INDEX))`。如所用的 `INDEX > 255`, 则要看它出现在第几个动态数据中, 并使 R 指向那个数据, 再求出相应的下标。则可以有 `WRITE(R↑.DATA(INDEX))`; 这实际上是把静态数组 ARY 与 A 类型的动态数据结合起来使用了。

当然, 这只是一个示意性的程序, 说明使用这种办法的可能性。如果解决实际问题, 还有一些细节问题要权衡考虑。如为什么选数组的下标为 $0 \cdots 255$? 选太大了、太小了都有什么问题? 如何尽可能缩短下标计算时间等等。

例 9-2 这个程序的功能是建立并遍历二叉树。这里说的二叉树, 是一种常用的数据结构, 它用递归术语定义, 并由递归算法加工, 它是由 n 个 ($n \geq 0$) 结点组成的有限集合。这个集合可以为空 ($n = 0$ 的情况), 也可以由一个根结点 最多再加上分别被称为左子树与右子树的互不相交的二叉树组成。

在这个程序中, 定义了四个过程, 其中过程 ENTER 的功能, 是从 TT.DAT 文件中读来原始数据, 并建立二叉树。其它三个过程 PREORDER, INORDER 和 POSTORDER 的功能分别是按前缀次序、中缀次序和后缀次序历遍 (走遍每一个结点) 建立起来的二叉树。这里说的前缀次序, 是指首先访问根, 再历遍左子树, 最后历遍右子树。中缀次序, 是首先历遍左子树, 再访问根, 然后历遍右子树。后缀次序, 是首先历遍左子树, 再历遍右子树, 最后访问根。

在 TT.DAT 文件中, 树是以前缀形式给出的, 其内容为: `ABC..DE..FG...HI..JKL..M..N..` 其中点表示空子树, 所以该二叉树的结构为:

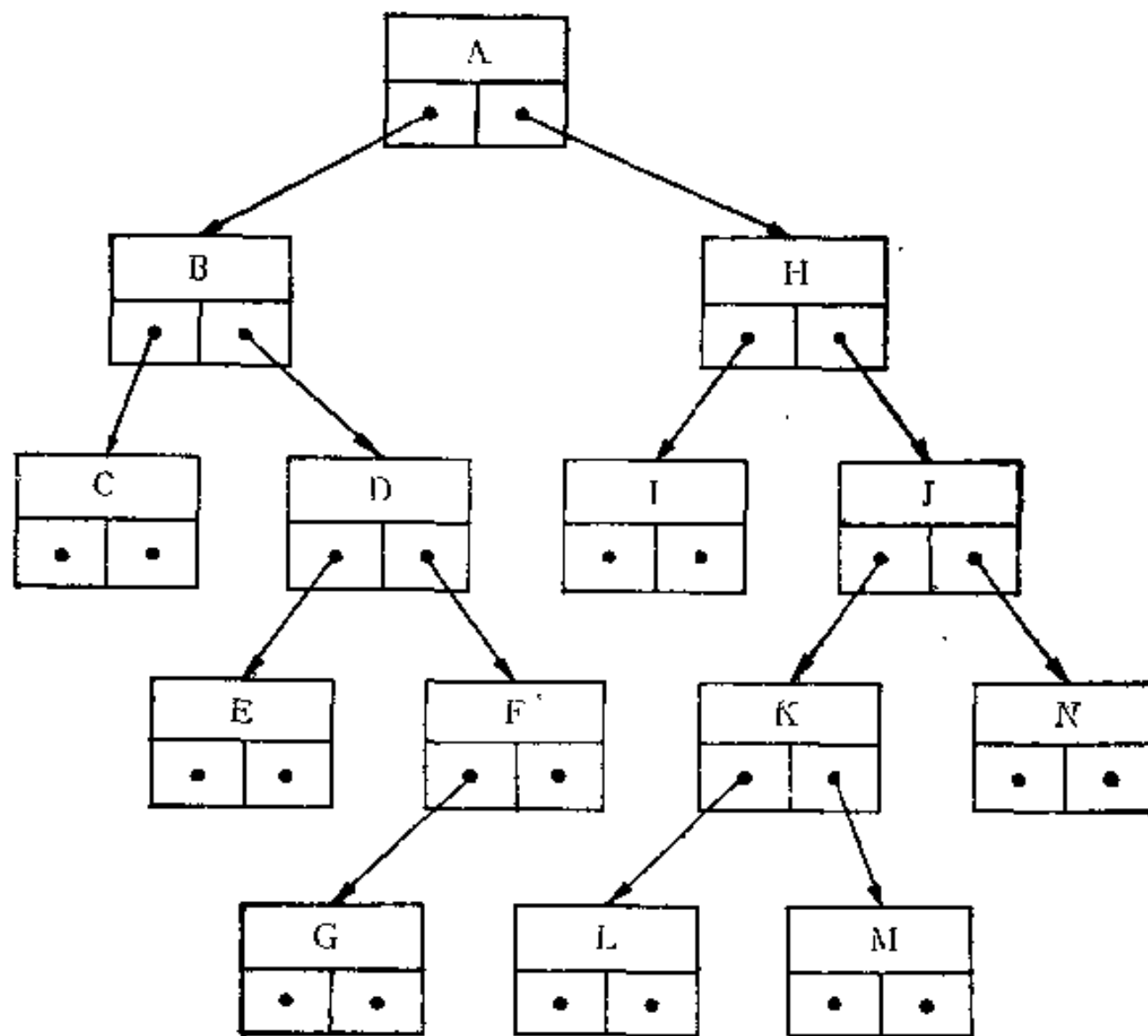


图 9.4-3 二叉树的结构

下面分析一下这个程序。PTR 是指针类型标识符, ROOT 为指针变量, 用它指向 NODE 类型的一个数据。NODE 是一个记录的类型标识符, 这个记录是由三个域组成的。其中有两

个域 LLINK 与 RLINK 为 PTR 类型，因此是指针变量。可以用它们分别指向左子树的节点与右子树的节点，第三个域 INFO 是字符变量，可以存放一个字符。

过程 ENTER 的功能是建立二叉数，具体办法是首先从源文件 TT.DAT 中读来一个字符数据，当这个字符不是 '·' 时（这表明又遇见了一个非空节点），就应处理这个结点。这时，首先用 NEW(P) 建立一个 NODE 类型的动态数据，并把读来的字符写进 P+.INFO 中去，这里的 P+ 是刚建立的这个动态数据本身的暂用名，所以 P+.INFO 就是这个动态变量的 INFO 域。这个程序的特点是递归调用，ENTER (P+.LLINK) 与 ENTER (P+.RLINK) 分别是建立相应的左子树与右子树。控制递归结束是用 CH='·' 这个条件实现的。此时执行 P:=NIL。

另外三个过程，分别是前缀次序、中缀次序与后缀次序遍历刚建立起来的二叉树。并输出每个结点的内容。这里要说明的是，每个动态数据中包括两个指针，是二叉树结构本身的要求，因为每个结点都要用它们指向自己的左子树与右子树。

这个程序看起来简单、清晰，但要真正弄清它的具体执行过程，是要仔细研究一下的。除了指针类型的说明与使用还不太熟悉外，主要困难在于过程的递归调用。这可以结合给出的二叉树具体结构进行分析与思考。我们准备把这个程序作更加详细的分析。

```

PROGRAM PNT04(INPUT, OUTPUT);
TYPE
  PTR=+NODE;
  NODE=RECORD
      LLINK, RLINK:PTR;
      INFO:CHAR
  END;
VAR
  ROOT:PTR;
  CH:CHAR;
  F1, F:TEXT;
  PROCEDURE PREORDER(P:PTR);
  BEGIN
    IF P<>NIL
    THEN
      BEGIN
        WRITE(F, P+.INFO);
        PREORDER(P+.LLINK);
        PREORDER(P+.RLINK)
      END
    END;
  PROCEDURE INORDER(P:PTR);
  BEGIN
    IF P<>NIL
    THEN
      BEGIN
        INORDER(P+.LLINK);

```

```

        WRITE(F, P+.INFO),
        INORDER(P+.RLINK)
    END
END,
PROCEDURE POSTORDER(P:PTR),
BEGIN
    IF P<>NIL
    THEN
        BEGIN
            POSTORDER(P+.LLINK),
            POSTORDER(P+.RLINK),
            WRITE(F, P+.INFO)
        END
    END,
PROCEDURE ENTER(VAR P:PTR),
BEGIN
    READ(F1, CH),
    WRITE(CH),
    IF CH<>'.'
    THEN
        BEGIN
            NEW(P),
            P+.INFO:=CH,
            ENTER(P+.LLINK),
            ENTER(P+.RLINK),
        END
    ELSE
        P:=NIL
    END,
BEGIN  (• MAIN PROGRAM •)
    RESET(F1, 'TT', 'DAT'),
    REWRITE(F, 'TT1', 'DAT'),
    ENTER(ROOT),      WRITELN,
    PREORDER(ROOT),   WRITELN(F),
    INORDER(ROOT),    WRITELN(F),
    POSTORDER(ROOT),  WRITELN(F),
    CLOSE(F)
END.
(•          TT.DAT          •)
  ABC..DE..FG...HI JKL M. N..
OUTPUT:
  ABC..DE .FG...HI JKL M. N..
(•          TT1.DAT         •)
  ABCDEFGHIJKLMN

```


CBEDGFAIHLKMJN
CHGFDBILMKNJHA
FIG PNT04 PROGRAM

例9-3 再看一个简化的大学新生入校登记表处理程序。这个程序的第一部分是造表登记每个人的情况。我们简单地把每个人的情况表示如下：姓名，年龄，性别，政治面目，入学总分等几项。每个人的情况是通过终端键盘送进去的。这一部分主要学习标准过程 NEW 的使用方法。第二部分是为了把登记过的全部内容，在终端显示器上显示出来。其要求是每个学生占一横行：这一部分内容主要是学习怎样使用已有的动态数据。第三部分是要在终端显示器上显示总分在 430 分以上的学生的姓名。这一部分也是学习怎样使用已有的动态变量，偏重学习怎样从全部动态数据中挑选出所要找的那些数据。

```
PROGRAM PNT05(INPUT, OUTPUT);
TYPE
  LINK='STUDENT',
  STUDENT=RECORD
    NEXT:LINK,
    NAME:ALFA,
    AGE:15..30,
    SEX:BOOLEAN,
    PYN:CHAR,
    SCORE:360..500
  END,
VAR
  P, Q, R:LINK,
  CH:CHAR,
  FIRST:BOOLEAN,
  F:TEXT,
BEGIN
  (* PART 1 *)
  FIRST:=TRUE; Q:=NIL; R:=NIL,
  REPEAT
    Q:=P; NEW(P);
    IF FIRST
    THEN
      BEGIN
        R:=P; FIRST:=FALSE
      END,
    IF Q<>NIL
    THEN Q^.NEXT:=P,
  WITH P+ DO
    BEGIN
      NEXT:=NIL,
      WRITE('STUDENT NAME:'), READLN(NAME),
      WRITE('STUDENT AGE:'), READLN(AGE),
```

```

    WRITE('STUDENT SEX:');      READLN(CH);
    IF CH='M'
        THEN SEX:=TRUE
        ELSE SEX:=FALSE;
    WRITE('STUDENT POLITY:');    READLN(PYN);
    WRITE('STUDENT SCORE:');    READLN(SCORE)
END;
WRITE('CONTINUE INPUT?');      READLN(CH);
UNTIL CH<>'Y';
(• PART 2 •)
REWRITE(F, 'PNT05', 'DAT');
WRITELN(F, '    NAME    AGE SEX PYN SCORE');
P:=R;
WHILE P<>NIL DO
    BEGIN
        WITH P+ DO
            BEGIN
                IF SEX
                    THEN CH:='M'
                    ELSE CH:='F';
                WRITELN(F, NAME, AGE:4, CH:3, PYN:4, SCORE:6)
            END;
        P:=P+.NEXT
    END;
(• PART 3 •)
WRITELN(F);
WRITELN(F, 'THE STUDENTS WITH OVER 430 SCORE ARE:');
P:=R;
WHILE P<>NIL DO
    BEGIN
        WITH P+ DO
            IF SCORE>=430
                THEN WRITELN(F, NAME);
            P:=P+.NEXT
    END;
CLOSE(F)
END.
INPUT:
ZHAO—YI  (• STUDENT NAME:      •)
22       (•    . .    AGE:      •)
M        (•    . .    SEX:      •)
P        (•    . .    POLITY:   •)
440      (•    . .    SCORE:    •)
Y        (• CONTINUE INPUT ?   •)

```

```

.
.
.
WU-FANG  (• STUDENT NAME:      •)
19        (•      . .      AGE:      •)
F         (•      . .      SEX:      •)
N         (•      . .      POLITY:    •)
425       (•      . .      SCORE:    •)
N         (• CONTINUE INPUT ?    •)

```

OUTPUT:

```
(•      PNT02.DAT      •)
```

NAME	AGE	SEX	PYN	SCORE
ZHAO-YI	22	M	P	440
QIAN-KONG	23	M	Y	415
SUN-ME	18	F	Y	435
LI-FANG	20	M	N	425
ZHU-MIN	21	M	Y	445
WU-FANG	19	F	N	425

THE STUDENTS WITH OVER 430 SCORE ARE:

ZHAO-YI

SUN-ME

ZHU-MIN

FIG PNT05 PROGRAM

由于这个程序比较简单，就不再给出它的框图，我们仅把几个有关问题说明如下：

1. 在 VAR 部分，我们说明了 P、Q、R 三个指针变量。从程序的执行部分可以看到，它们各有自己的用途。R 用作为指向头一个动态数据的指针；P 用来作 NEW 过程的实际参数，动态地建立每个数据。在新建立每一个动态数据时，用 Q 指向在这个数据前边建立的那个数据。这样就可以用 Q·NEXT 指向这个新建立的数据，换言之，要把新建立的数据的首地址写入前一个数据中的指针里。我们可以把 R、P、Q 的作用表示如下：

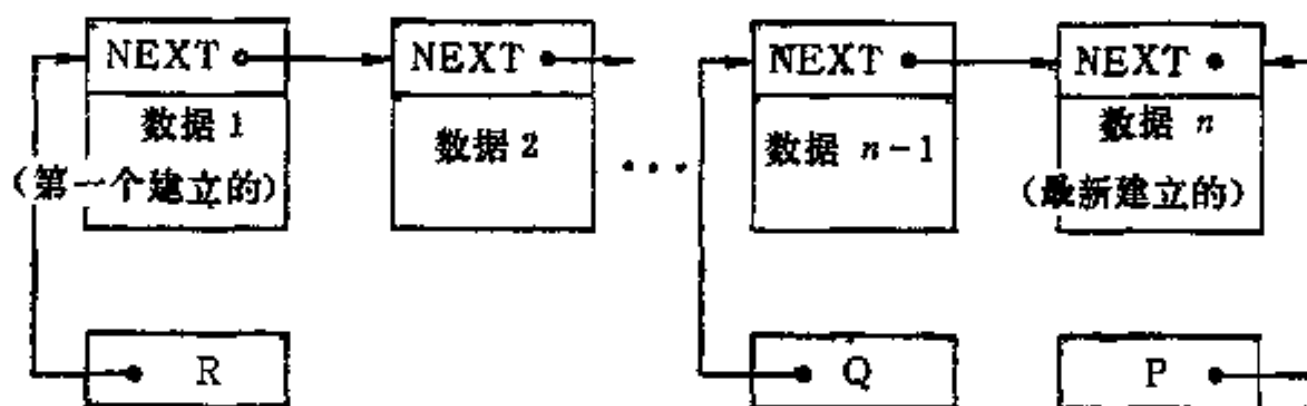


图 9.4-4 指针 R、P、Q 的作用

2. 在 VAR 部分说明了一个布尔型变量 First，其用途是为了区别所建立的动态变量是否为头一个。之所以要把头一个与后面的其它数据区分开，有以下两个原因：

a. 如果是头一次用 NEW(P) 过程，则要把 P 的值赋给 R，以便登记下这一串动态数据的首址。以后再查找这些数据，就从 R 指向的那个数据查起。

b. 如果不是头一次调用 NEW(P)过程, 则要把此时的 P 的值赋给它前面一次建立的那个数据 (此时由 Q 指向它) 的 NEXT 域。

上面两点实现起来并不难, 先设置 R 为 NIL, Q 为 NIL, 并让 First 为 TRUE。当用 REPEAT 登记每个学生的情况时, 则看一下 First 的值, 若为 TRUE, 则表明是头一次执行 NEW(P), 就要把 P 的值赋给 R, 并让 First 变为 FALSE。查 Q 是否为 NIL, 也是为了区分是否头一次调用 NEW(P), 因为程序开始时使 Q 为 NIL, 而在建立好第一个数据之后, 当执行过 $Q:=P$ 语句之后, Q 就不再为 NIL 了。

3. 在 VAR 部分说明一个字符型变量 CH, 在登记学生性别时要用到它, (在判 REPEAT 语句是否结束时, 也要用到 CH)。这是因为性别 SEX 本身是布尔型变量, 无法通过 READ 语句输入它的真 (男性) 假 (女性) 值。此时我们就通过输入的字符来表明男女。打字符 M 表示男, 否则为女: 即这里是根据输入的字符是否为 M, 来登记 SEX 域的值。

4. 我们用 REPEAT 语句建立每一个动态数据, 每建完一个数据, 程序要问你还要继续登记, 如果是, 则输入字符 Y, 就继续重复, 否则表示登记完毕, 退出循环, 这是通过判 CH 是否为字母 Y 来实现的。从这里又一次看到, 动态数据的个数是可以改变的, 到底建立多少个数据, 仅取决于用户何时不回答 Y。

5. 在程序的第二部分, 为了把登记好的情况从终端上显示出来, 只要从第一个数据起, 依次按给定格式输出就行了。具体作法是, 首先把 R 的值赋给 P, 并且判 P 是否为 NIL, 以决定是否应结束输出。因为最后一个学生的数据中的 NEXT 域总是 NIL 值。所以, 执行语句 $P:=P \cdot \text{NEXT}$ 之后, 若 P 为 NIL 值, 则结束输出。

6. 程序的第三部分与第二部分中的区别仅在于, 只把总分在 430 分以上的学生的名字输出来, 既不是每个 430 分以上的学生的全部情况, 也不是所有学生的名字。这是使用动态数据时常常采用的一种方式。

例9-4 这是一个已经简化了的示意性编译程序的例子。这个程序可以对类型为 .CPL 的某些文件进行处理。

这里是文件 ABC.CPL 的一个例子:

```
PROGRAM ABC;
CONST
  A=3;   B=14;   C=25;
VAR
  D, E, F, G, INTEGER;
BEGIN
  D:=10-B;
  E:=12+E;
  F:=-1+234;
  G:=A+F;
END.
```

必须强调指出这个程序与真正的编译程序既有某些相似之处, 又有着明显的区别。要知道, 我们在这个程序中, 主要着眼点是说明应怎样建立与使用动态数据, 而不是编译程序本身的工作原理。

下面就比较详细地分析一下这个程序。

```

PROGRAM PNT06(INPUT, OUTPUT);
LABEL
  1, 2, 3, 4, 5, 6;
TYPE
  DATATYPE=(CONSTANT, VARIABLE),
  ADRS=0.177777B;
  LINK=*DATA;
  DATA=RECORD
    NEXT:LINK;
    NAME:ALFA;
    CASE DTYPE:DATATYPE OF
      CONSTANT:(VALUE:INTEGER);
      VARIABLE:(ADDRESS:ADRS)
    END;
  ERRMESSAGE=ARRAY (1..30) OF CHAR;
VAR
  LEN, I, J, K, L:INTEGER;
  CH, CH1:CHAR;
  A, FILENAME:ALFA;
  DIGITS:SET OF '0'..'9';
  LETTERS:SET OF 'A'..'Z';
  P, Q, R:LINK;
  F, FOUT:TEXT;
  PROCEDURE NEXTCH;
  BEGIN
    IF NOT EOF(F)
    THEN
      IF NOT EOLN(F)
      THEN READ(F, CH)
      ELSE
        BEGIN
          READLN(F);
          CH:=F^1;
        END
      ELSE
        BEGIN
          WRITELN(' END OF FILE TO COMPILE'); GOTO 2
        END
      END;
  END;
  PROCEDURE INSYMBOL;
  BEGIN
    K:=0;
    NEXTCH;
    WHILE CH=' ' DO NEXTCH;

```

```

IF CH IN LETTERS
THEN
BEGIN
    K:=1;      I:=0;
    REPEAT
    IF I<10
    THEN
    BEGIN
        I:=I+1; A(I):=CH; NEXTCH
    END
    ELSE NEXTCH;
    UNTIL NOT (CH IN LETTERS) AND
    NOT (CH IN DIGITS);
    WHILE I<10 DO
    BEGIN
        I:=I+1; A(I):=' '
    END;
END
ELSE
IF CH IN DIGITS
THEN
BEGIN
    K:=2;      J:=0;
    REPEAT
        J:=10*J+ORD(CH)-ORD('0');
    NEXTCH
    UNTIL NOT (CH IN DIGITS);
END
END;
PROCEDURE SKIP (CH1:CHAR);
BEGIN
    WHILE CH<>CH1 DO NEXTCH;
    NEXTCH
END;
PROCEDURE WRTERROR(ERR:ERRMESSAGE,CHR:CHAR);
BEGIN
    WRITELN(ERR); SKIP(CHR)
END;
PROCEDURE CHERCH;
BEGIN
    P:=Q;
    WHILE P<>NIL DO
    IF A<>P^NAME
    THEN P:=P^NEXT

```

```

        ELSE EXIT
    END,
BEGIN      (• MAIN PROGRAM •)
    (• PART1 — INPUT THE NAME OF FILE TO COMPILE •)
    DIGITS := {'0'..'9'} ,
    LETTERS := {'A'..'Z'} ,
    REPEAT
        WRITE('FILE NAME TO COMPILE'),
        READLN(FILENAME),
        RESET(F, FILENAME, 'CPL', LEN),
        IF LEN=-1
            THEN WRITELN('THIS IS NO THE FILE INDICATED,')
        UNTIL LEN<>-1,
    (• PART2 — TREAT THE PROGRAM HEAD •)
    INSYMBOL,
    IF A<>'PROGRAM'
        THEN WRITELN('MISSING "PROGRAM"'),
    INSYMBOL,
    SKIP(',');
    (• PART3 — TREAT CONST DECLARATION •)
    Q:=NIL, P:=NIL,
    INSYMBOL,
    IF A='CONST'
        THEN
            BEGIN
                INSYMBOL,
                REPEAT
                    IF K<>1
                        THEN
                            BEGIN
                                WRERROR('AN IDENTIFIER EXPECTED', ', ', ')',
                                GOTO 3
                            END,
                UNTIL K=1,
                CHERCH,
                IF P<>NIL
                    THEN
                        BEGIN
                            WRITELN(A, ' DECLARE TWICE'),
                            SKIP(',');GOTO 3
                        END,
                SKIP('=');
            INSYMBOL,
            IF K<>2
                THEN

```

```

      BEGIN
        WRERROR('A NUMBER EXPECTED', ' ', ' '),
        GOTO 3
      END,
    NEW(P),
    WITH P+ DO
      BEGIN
        NEXT:=Q, NAME:=A, VALUE:=J, DTYPE:=CONSTANT
      END,
      Q:=P, SKIP(' '),
3
      INSYMBOL
      UNTIL A=VAR
    END,
    (* PART 4 -- TREAT THE VAR DECLARATION *)
    IF A='VAR'
    THEN
      BEGIN
        INSYMBOL,
        REPEAT
          IF K<>1
          THEN
            BEGIN
              WRERROR('AN IDENTIFIER EXPECTED', ' ', ' '),
              GOTO 4
            END,
          CHERCH,
          IF P<>NIL
          THEN
            BEGIN
              WRITELN(A, 'DECLARE TWICE'), SKIP(' '),
              GOTO 4
            END,
          NEW(P),
          WITH P+ DO
            BEGIN
              NEXT:=Q, NAME:=A, DTYPE:=VARIABLE,
              ADDRESS:=0
            END,
            Q:=P,
4
            INSYMBOL,
            IF (CH=' ') AND (N=0)
            THEN

```



```

        BEGIN
            SKIP(' '), INSYMBOL
        END,
        UNTIL CH=' ',
        SKIP(' '),
        INSYMBOL,
        END,
IF A<>'INTEGER'
    THEN WRITELN('VARIABLE DATA TYPE ERROR'),
    SKIP(' '),
    INSYMBOL,
(* PART 5 -- TREAT THE PROGRAM EXECUTION *)
IF A='BEGIN'
    THEN
        BEGIN
            INSYMBOL,
            REPEAT
                CH1:=' ',
                CHERCH,
                IF P=NIL
                    THEN
                        BEGIN
                            WRITELN(A, ' IS AN UNDEFINED SYMBOL '),
                            SKIP(' '),
                            GOTO 5
                        END,
                IF P.DTYPE=CONSTANT
                    THEN
                        BEGIN
                            WRITELN('CANNOT BE CONSTANT '),
                            SKIP(' '), GOTO 5
                        END
                ELSE R:=P,
                SKIP(' '), SKIP('= '), L:=0,
                INSYMBOL,
1:
                LEN :=0,
                IF K=0
                    THEN
                        BEGIN
                            IF CH='-'
                                THEN CH1:='- '
                            ELSE
                                IF CH='+'

```

```

    THEN CH1 := '+'
    ELSE
    IF CH=','
    THEN GOTO 8
    ELSE
    WRITELN('SYNTAX ERRORS IN THIS STATEMENT ')
END
ELSE
BEGIN
    IF K=2
    THEN LEN := J
    ELSE
    BEGIN
        CHERCH,
        IF P <> NIL
        THEN
        BEGIN
            IF P.DTYPE=CONSTANT
            THEN LEN := P.VALUE
            ELSE LEN := P.ADDRESS
        END
        ELSE
        BEGIN
            WRITELN(A 'IS AN UNDEFINED SYMBOL'),
            SKIP(',') , GOTO 5
        END
    END
    IF CH1='-'
    THEN L := L - LEN
    ELSE L := L + LEN,
    CH1 := ' ';
END
INSYMBOL GOTO 1,
6.
R. ADDRESS:=L. SKIP(',')
5.
INSYMBOL
UNTIL A='END' ,
SKIP(' '),
END,
(* PART 2 — OUTPUT THE RESULT OF COMPILER *)
REWRITE(FOUT, 'ABC', 'DAT'),
WRITELN(FOUT, ' ',10,'ABC.DAT'),
WRITELN(FOUT, ' THE RESULT OF COMPILER:'),

```

率

```

P:=Q,
WHILE P<>NIL DO
  BEGIN
    WITH P* DO
      BEGIN
        WRITE(FOUT, NAME),
        IF DTYPE=CONSTANT
          THEN WRITELN(FOUT VALUE)
          ELSEF WRITELN(FOUT, ADDRESS)
      END;
    P:=P* NEXT
  END;
CLOSE(FOUT);
2:
END
(•      ABC CPL      •)
PROGRAM ABC;
CONST
  A=3;  B=14;  C=25;
VAR
  D, E, F, G: INTEGER;
BEGIN
  D:= 10 - B;
  E:= 12 + E;
  F:= - 1 + 234;
  G:= A+F;
END
OUTPUT:
(•      ABC DAT      •)
THE RESULT OF COMPILER:
G                                236
F                                233
E                                12
D                                -4
C                                25
B                                14
A                                3
      FIG PNT06  PROGRAM

```

1. 先对文件 ABC.CPL 的内容说明如下:

(1) 这个文件是我们例证程序的处理对象。所以它的基本格式应与 PASCAL 源程序类似 并且 PROGRAM, CONST, VAR, BEGIN, END 和 INTEGER 几个字 为保留字, 它们的用法与它们在真正的 PASCAL 源程序中的用法相同。

(2) 其它的符号, 如程序名 ABC; 常量名 A, B, C; 变量名 D, E, F 等均为用户定

义的符号。它们可以由若干字母与数字字符组成，但头一个字符必须为字母。这与 PASCAL 规定的组成标识符的规则相同。如果它们由多于10个的字母与数字字符组成，仅前十个符号有效。

(3) 我们的程序不对这里说明的常量名与变量名的个数规定任何限制，但如果遇到同一名字被重复说明（两次或两次以上）的情况，它将查出这类错误。如果在 BEGIN 之后的执行部分，出现没有说明过的符号，它也能查出这类错误。

如果把程序中的 ABC.CPL 修改如下，则对它编译的结果会如(2)、(3)指出的那样进行处理和查出相应的错误。

(• ABC.CPL •)

PROGRAM ABC ;

CONST

A123456789P = 3 ;

A123456789K = 1 ;

B = 14 ; C = 25 ;

VAR

D, E, F, G: INTEGER;

BEGIN

D := 10 - B;

E := 12 + E;

F := -1 + 234;

G := A + F;

END .

THE RESULT OF COMPILER FOR ABC CPL:

A123456789 DECLARE TWICE

A IS AN UNDEFINED SYMBOL

(• ABC.DAT •)

THE RESULT OF COMPILER:

G	0
F	233
E	12
D	-4
C	25
B	14
A123456789	3

(4) 常量的值，为简化起见，规定只取正整数，同样，变量说明中，只能说明 INTEGER 类型的变量。

(5) 还是为了简便，这里规定可以进行的运算只有加法与减法，当然，只可以把运算的结果赋值给某个变量，而不能赋值给一个常量（如 A := E - C 就是一个错误）。

如果读者按上述说明重写 ABC.CPL 文件的内容，我们给出的程序仍能对其进行正确的处理。

2. 关于例证程序本身，因为这个程序本身比较复杂，又涉及到有关编译技术中的一些问题。对于某些只搞过硬件的读者来说，不容易通过自学全部掌握。为此，就FIG PNT06 PROGRAM所示的程序，进一步解释如下：

(1) 在这个程序的 TYPE 说明中, 把 DATA 说明为一个记录。它由两个固定域与一个变体域组成。一个固定域为 NEXT, 用于勾链每个动态变量, 另一个固定域为 NAME, 用来存放一个常量或变量的名字。变体域, 可能是一个 CONSTANT, 这时就要有一个单元存放这个常量的数值。也可能是个变量 (在我们这儿只能为整数型变量); 这时就要有一个单元存放这个变量所在的地址。为了简便, 我们这里仍写它为 ADDRESS, 但并不用它真正存放变量地址。而是用它存放变量本身的值。这是因为我们的所谓的编译程序并没有代码生成部分, 不产生 OBJECT 文件, 所以也未曾为变量分配地址。从这个角度讲, 我们的程序只是处理 PASCAL 程序的说明部分的简化形式。

(2) 再看这里说明的几个过程:

NEXTCH 用来从被处理的文件中取来一个字符，取来的字符在变量 CH 中。

INSYMBOL 完成词法分析功能，它从被处理的文件中读来一个 SYMBOL，它可以是一个标识符。如 PROGRAM, ABC, 结果放在 A 中。如果这个符号由 10 个以上的字母数字字符组成，仅前十个有效。如果它本身少于 10 个符号，则不足部分补上空格字符。因此，在这种情况下，取来的一定是 ALFA 类型。这时让 K 为 1，表示本次取来的是一个标识符。否则，它也可能是个数字。其结果放在 J 中，并用 K 等于 2 来表示这种情况。如果不属于上述两种情况，则让 K 等于 0。此时 CH 中存放着一个专用字符。在我们的例子中，它可能是 ' ; ' , ' / ' , ' : ' 或 ' = '。

SKIP 过程完成跳过某个字符的操作。

CHERCH 过程用来在建立起来的动态数据中查找某个名字。如果查到了, B 为 TRUE, 否则为 FALSE。

(3) 在程序的执行部分, 有以下几个问题要说明:

在PART 1中,从第四行开始的 REPEAT 语句的作用是把要处理的文件名读进来,如果在盘上找不出你给出名字的那个文件,则程序会提示你:

THIS IS NO THE FILE INDICATED.

这时你可以再给出一个文件名，直到能找到你给出的文件名的那个文件为止。

再下面的几行，处理程序首部 (PART2):

在 PART3 与 PART4 中, 通过标准过程 NEW 建立 DATA 类型的动态数据, 即登记常量与变量的名称及数值, 并正确地把它们勾链起来。

这里要特别强调说明几个问题：在处理常量说明时，是把它名字与数值一次登记好了；而在处理变量说明时，真正的编译程序应为每个变量分配地址，并把变量名字与它的地址值登记好；但在我们的程序中，并没有给变量分配地址，所以，正如前面已提到的，我们是改用 ADDRESS 存放变量的值。但这时变量的值尚未确定，所以先一律填零值。另外一个问题是，在处理每个常量与变量说明时，总是先到已建立的动态数据中查一查，看是否前面已用过这个名字，如果前面已有了，则现在遇到的是重复说明，这是一个错误，要指明这类错误。如果没有查到这个名字，则说明这是第一次遇到，就要登记它。由此可见，在建立动态数据的过程中，就会用到那些已建立起来的动态数据。第三个问题是处理动态变量的变体部分。在处理常量说明时，DTYPE 域为 CONSTANT，并且，用一个 VALUE 单元记

忆一个常量的数值：在处理变量说明时，DTYPE 域为 VARIABLE，则应该用 ADDRESS 单元记忆变量地址，我们这里却用这个单元记忆变量本身的数值了。

在处理 BEGIN 后面的语句时，(PART5)，主要问题是查一查，遇到的每个标识符(常量与变量名字)是否都在前面的说明部分说明过。这就要查找建立的动态数据，如果前面说明部分未说明过这个标识符，则它就是一个未定义的 SYMBOL。下一个问题是检查出现在赋值符号(:=)左面的名字是否是变量名，若为常量名，则是一个错误。第三个问题是，为了执行每个赋值语句，真正的编译程序应给出一系列机器指令，实现对赋值号右边的表达式的运算。我们这里则是找到每个量(常量或变量)的值。再由我们的程序按表达式中给出的“+”或“-”号完成所要求的运算操作与结果的赋值操作。

程序的最后一部分，也就是处理完整个被处理的程序后，再把每个变量的名字和它的数值，每个常量的名字和它的数值都在终端上显示一遍。

在有了上述说明之后，读者应有能力自己看懂这个程序。这个程序涉及到动态变量中的变体部分：应弄清它的说明方式与用法。这个程序也碰到某些编译程序中用到的概念与技术，这也许会向那些从来未接触过编译程序的同志，提供一点启蒙性的概念知识。顺便提一句，用 PASCAL 语言写这个语言本身的编译程序，是目前普遍采用的一种手段。

§5 PASCAL程序与MACRO一级的程序衔接

OMSI PASCAL-1语言，允许在 PASCAL 源程序中间插入 MACRO 语言的源程序段。具体方式是把要插入的程序段，以一种特殊的注释形式写在 PASCAL 源程序中。

一、一个例证程序：

```
PROGRAM PNT07(OUTPUT),
TYPE
  L = 'INTEGER',
  K = 'L',
VAR
  P,R:L,
  M,N:K,
  I,J:INTEGER,
BEGIN
  I := 5,      J := 10,
  P := @I,     R := @J,
  M := @P,     N := @R,
  WRITELN(I:6,P:6,M:6,N:6),
  (• $C , INSERTED MACRO SECTION
MOV      0(5), -(6)
CLR      -(6)
JSR      %7, $B24
MOV      2(5), -(6)
CLR      -(6)
JSR      %7, $B24
JSR      %7, $B36
```

```

MOV      4(5),-(6)
CLR      -(6)
JSR      %7,$B24
MOV      6(5),-(6)
CLR      -(6)
JSR      %7,$B24
JSR      %7,$B36
*)
WRITELN(J:6,R↑:6,N↑↑:6)
END .
OUTPUT:
      5      5      5
      1938      1940
      1930      1932
10     10     10
FIG    PNT07    PROGRAM

```

下面详细说明一下这个程序。

在类型定义部分，定义了L与K两个指针类型标识符。请大家注意， $K = \uparrow L$ 这一说明。

在程序执行部分，语句 $P := @I$ ，是把I这个变量的地址传送给指针P（检查TYPE与VAR说明部分可以看到，P是指向一个整型变量的指针）。这里用到的符号“@”，是OMSI PASCAL语言给出的使用单操作数的一个操作符，被称为地址操作符，可以用它把一个变量的地址传送给相应类型的指针。因此，在 $P := @I$ 语句执行过之后，P中放的是变量I的地址。此时， P^\uparrow 就表示I变量本身了。 $M := @P$ ，是把指针P的地址赋给M，那么 M^\uparrow 就表示指针P， $M^{\uparrow\uparrow}$ 就表示P所指向的那个变量，也就是I了。这可以通过WRITELN(I, P^\uparrow , $M^{\uparrow\uparrow}$)语句的输出结果得到证实，这一输出结果为三个整数值5。

这里请注意，不能把 $M^{\uparrow\uparrow}$ 写成 $(M^\uparrow)^\uparrow$ ，这种写法不符合PASCAL语法规定，因为小括号只能用在表达式中， $M^{\uparrow\uparrow}$ 不是一个表达式。

在(*\$C与*)之间是插入的一段MACRO程序，在注释中的前两个字符\$C有特定的含意，它表示下面是插入的一段MACRO程序。这一段程序的功能是把I, J, P和R的地址打印出来。在PASCAL语言一级的程序中，只能输出指针所指向的变量的内容，如 P^\uparrow , R^\uparrow , $M^{\uparrow\uparrow}$, $N^{\uparrow\uparrow}$ ，而不能直接输出指针变量本身。如WRITE(P, M^\uparrow)等。如果确实需要输出指针本身的内容，即它们中存放的地址，就只能到MACRO语言一级的程序中进行。在我们这里的这一段MACRO程序中，用到了PASCAL库程序中的两个标准子程序\$B24和\$B36，它们的功能分别是输出已经给出地址的一个整数和执行输出换行，对用到的其它的指令我们不予说明，请参考PDP-11/03机的指令系统。

二、PASCAL程序与插入的MACRO程序衔接中的三个问题。

1. 通讯中的变量识别问题

在用PASCAL语言与用MACRO语言写的两部分程序之间怎么进行通讯呢？即怎么在MACRO语言的程序中使用上PASCAL程序中说明的变量呢？有三种解决办法：

第一种办法，通过使用变量名使用变量。对整型的全程变量来讲，我们只要在变量名之

后写上 (5) (意思是用 R_5 指明它) 或 (R_5), 就可以把它用作为指令中的一个操作数了。对实型数, 字符型数, 构造类型数据, 需要程序员在自己的这部分程序中, 通过妥善地处理相应地址关系来使用这些数据。

第二种办法, 通过确定的变量地址来使用变量。在 OMSI PASCAL-1 语言中规定, 可以在说明变量时, 为某些变量指定确定的 (用绝对地址表示) 地址。如:

```
VAR
  I   ORIGIN 1200:INTEGER;
  AR  ORIGIN 2300(1..20) OF CHAR;
```

这就表明, 要把整型变量 I 分配在绝对地址为 1200 (字地址) 的内存单元中, 而把 20 个字符组成的一个字符串分配在从绝对地址为 2300 开始的 10 个存储字中。这样在 MACRO 程序段中, 可以通过这些已知的地址使用 I 与 AR 两个变量。

第三种办法是利用指针变量, 详细内容请参见 PROGRAM PNT07 及那里的说明。

2. 有关寄存器内容的保留与恢复问题

在对 PASCAL 语言进行编译的过程中, 要用到 $R_0 \sim R_6$, 以及 R_5 (SP), R_7 (PC) 八个通用寄存器。当遇到插入的一段 MACRO 程序时, 它就原封不动地把它插在编译程序产生出来的 MACRO 目标程序中, 这就可能出现两部分之间在使用寄存器 $R_0 \sim R_6$ 方面的冲突。为此, 在插入一段 MACRO 程序之前, 首先应把 $R_0 \sim R_6$ 的内容暂时保存起来, 插入的 MACRO 程序段结束前, 再恢复它们。这就像计算机处理中断时的保存与恢复现场一样。

3. 数制 (RADIX) 配合问题

OMSI PASCAL-1 编译程序产生的 MACRO 类型文件, 用的是十进制数, 而写 MACRO 程序习惯上用八进制。对插入的 MACRO 程序段, 如果不对数制做任何说明, 它就沿用编译程序开始时约定的十进制, 如果想用八进制, 则首先要用 `*RADIX 8` 约定所用数制, 并在 MACRO 程序段结束时, 用 `*RADIX 10` 恢复十进制, 保证后面的编译结果的正确性。

在进行了这些说明之后 看几个简单的程序:

```
PROGRAM PNT08(OUTPUT);
VAR
  I : INTEGER;
BEGIN
  I = 9;
  (* $C
    MOV      I(5), R0
    ADD      #10, R0
    MOV      R0, I(5)
  *)
  Writeln('I=', I:5)
END .
OUTPUT:
I =      19
FIG PNT08 PROGRAM
```

PNT08 程序中插入的一段 MACRO 程序实现的, 是把 I 的值 (这里用 I(5) 表示)

传送到 R0 中，再加上 10（这是十进制的 10，因为没有特别说明数制，则沿用 PASCAL 编译程序约定的十进制），然后送回 I 单元去。所以最后的输出结果应为 19。

```

PROGRAM PNT09(OUTPUT),
VAR
  ! ORIGIN 1200 : INTEGER,
BEGIN
  I := 9,
  ( * $C
    .RADIX 8
    MOV 1200 R0
    ADD #10, R0
    MOV R0, 1200.
    .RADIX 10
    * )
  WRITELN('I = ', I:5)
END .
OUTPUT:
I = 17

```

FIG PNT09 PROGRAM

PNT09程序与 PNT08相比有两点区别，一个在说明 I 变量时，为它指明了绝对地址，所以在 MACRO 程序段中，是通过这个绝对地址访问 I 变量的；二是 MACRO 程序段中用的是八进制，所以 ADD *10, R0 指令中的 10 是八进制数，它表示的是十进制的 8，所以 WRITELN (I) 输出的结果是 17，而不是上一个程序的 19。

```

PROGRAM PNT10(OUTPUT),
TYPE
  L = 'INTEGER,
  K = *L,
VAR
  P, R : L,
  M, N : K,
  I, J : INTEGER,
BEGIN
  I := 5,          J := 10,
  P := @I,         R := @J,
  M := @P,         N := @R,
  WRITELN(I:6, P:6, M:6);
  ( * $C , THEN FOLLOWING IS AN INSERTED MACRO SECTION
    MOV P(5), --(6)
    CLR --(6)
    JSR %7, $B24 , OUTPUT ADDRESS OF I
    MOV M(5), --(6)
    CLR --(6)
    JSR %7, $B24 , OUTPUT ADDRESS OF P
  )

```

```

        JSR          %7, $ B36          ,LINE FEED
    • )
    WRITELN(J:6,R↑:6,N↑↑:4)
END .
OUTPUT:
        6          5          6
          1904          1896
       10         10         10
    FIG    PNT10    PROGRAM

```

我们可以对本节开始时给出的程序改写一下，即在插入的 MACRO 程序段中通过变量访问这些变量，如 PNT10 程序所示。这段程序的功能是输出 I 与 P 的地址。由于输出完这两个地址后没用 \$ B36 进行输出换行操作，所以整个程序的最后一个 WRITELN 语句中的三个数，也被输出在这同一行上。

```

    PROGRAM PNT11(OUTPUT);
    TYPE
        L = ↑A ;
        A = ARRAY (1..10) OF INTEGER;
    VAR
        I : INTEGER ;
        LK : L ;
        AR : A ;
    BEGIN
        FOR I := 1 TO 10 DO AR (I) := I ;
        LK := @AR ;
        (• $ C, THIS PART IS USED TO CALCULATE THE SUM OF AR'S ELEMENTS
            MOV      LK(5),RO  ;MOVE THE AR'S ADDRESS INTO RO
            CLR      R2
            MOV      #10,R3    ;MOVE ADD OPERATION TIMES INTO R3
        LOOP: ADD   (0),R2    ;CALCULATE THE SUM
            ADD      #2,R4     ;MODIFY THE ADDRESS POINTING TO THE
                               NEXT ELEMENT
            DEC      R3        ;CHECK OPERATION TIMES
            BNE      LOOP     ;EXIT LOOP IF FINISH ED THE OPERATION
            MOV      R2,I(5)   .MOVE THE SUM TO I VARIABLE
        • )
        WRITELN('I= ',I:5)
    END .
OUTPUT:
    I =      65
    FIG    PNT11    PROGRAM

```

PNT11 程序的功能是先对数组 AR 的每个元素赋值，插入的一段 MACRO 程序求这个数组所有元素的和。

最后一个 PASCAL 语句输出结果, 应为 55 (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)。

看最后一个例证程序 PNT12, 用插入的宏汇编处理浮点运算。

```
PROGRAM PNT12(OUTPUT);
VAR
  R1,R2  : REAL;
BEGIN
  R1 := 12.50;
  R2 := 32.74;
  ( • $C
  MOV      2(5), -(6)
  MOV      @%5, -(6)
  MOV      6(5), -(6)
  MOV      4(5), -(6)
  FADD      %6
  MOV      (6)+, @%5
  MOV      (6)+, 2(5)
  • )
  Writeln('R1=', R1);
  Writeln('R2=', R2);
END.
OUTPUT:
R1  = 4.524000E+01
R2  = 3.274000E+01
FIG  PNT12  PROGRAM
```

插入的这段 MACRO 程序实现的是 $R2 + R1$, 结果保存在 $R1$ 中, 可以通过 Writeln (R1) 语句的输出结果校验一下。存放 $R2$ 的存储单元的内容未变, 依然是 32.74, 只是书写格式变了一些。

本章小结

这一章的重点, 是讲解通过指针建立并使用动态数据的方法。动态数据是一种常用的数据结构, 应了解它的特点与用法。

关于指针本身的有关概念, 应能比较好地懂得 'OBJECT, P 与 P' 等符号的含义。在指针的应用方面, 要了解它的两种主要用法。一是通过地址操作符 @, 把 PASCAL 中的某些全程变量的地址, 传送给 MACRO 一级的系统子程序, 或插在 PASCAL 程序中的 MACRO 程序段; 二是通过指针建立并应用动态数据。相比之下, 这第二种用法是更重要的, 应很好地掌握这一部分的内容。

本章习题

1. 输入一系列整数，当遇到 0 时停止，建立并输出偶数链表和按从小到大排列的奇数链表。
2. 通过终端输入某班 N 个学生的情况（N 也由终端键盘输入）。用指针类型在磁盘建立学生情况表的文件 S1.DAT，包括姓名，年龄，性别和成绩。
3. 从磁盘调出文件 S1.DAT，按学生的年龄排队（从小到大）形成一个新的表，以 S2.DAT 为文件名，存放在磁盘中。如果插入一个学生（确定的姓名，年龄，性别和成绩）到适当的位置，形成新表，文件为 S3.DAT。
4. 已知一个链表有十个结点 A1 A2, ……A10 分别存放 X1, X2, ……X10 等十个整数。现在要求由整数 Y2 代替 X2, Y4, Y5 插入 A4 和 A5 结点之间，并删除结点 A8，形成一个十一个结点的链表。试编程序实现上述功能。

第十章 PASCAL 程序的建立、运行与调试

本章内容，是根据 PDP-11/03 计算机系统和在这个系统中运行的 OMSI PASCAL-1 语言来编写的。如果读者使用其它系列计算机，本章的某些细节内容不能完全照搬。请用户参照自己计算机系统手册中的具体说明。不过，本章的基本概念、方法、步骤及总的思路，在各种类的计算机系统中是相同的或类似的。

本章主要介绍四项内容：

- 一、建立 PASCAL 源程序的方法。重点介绍如何使用在 PDP-11/03 系统中运行的文本编辑与修改程序 TECO；
- 二、在 PDP-11/03 系统中，运行一个 OMSI PASCAL-1 源程序的步骤与方法。主要学习如何使用有关的键盘命令；
- 三、检查 PASCAL 源程序语法错误并进行相应修改的方法。主要介绍如何通过编译程序建立源程序的列表（或称清单）文件，并利用这一文件修改源程序中的各项语法错误；
- 四、控制目标程序的运行，检查出程序中可能出现的逻辑上的错误的方法。主要讨论如何正确使用 OMSI PASCAL-1 语言的辅助调试程序 DEBUGGER。

在具体讲解本章内容之前，先简明地介绍最简单的 PDP-11/03 计算机系统的组成和常用的某些符号。

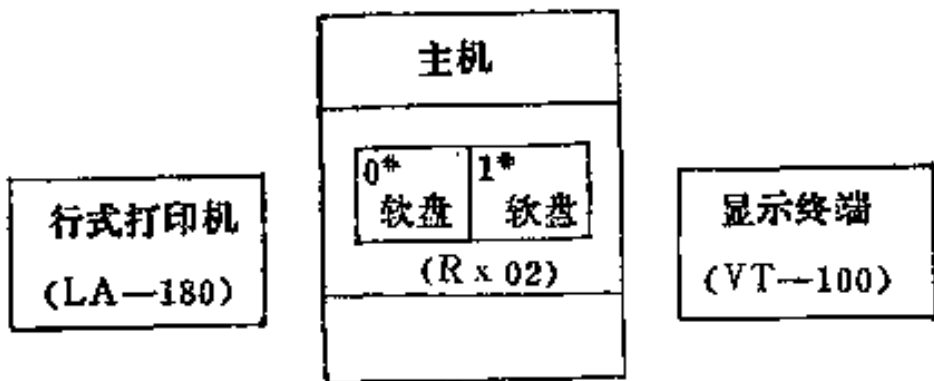


图 10.0-1 PDP-11/03 计算机系统的组成

一个最简单的 PDP-11/03 系统的组成情况，如图 10.0-1 所示：

在这个系统中，软磁盘的设备型号是 R × 02，它分为 0* 软盘和 1* 软盘，行式打印机的型号是 LA-180，显示终端为 VT-100。它们的设备名又分物理名称和逻辑名称，列表如下。

PDP-11/03 计算机系统设备名称			
		物 理 名 称	逻 辑 名 称
软 磁 盘	0* 软 盘	DY0：，或 DY：	SY：
	1* 软 盘	DY1：	DK：
行 式 打 印 机		LP：	LP：等
显 示 终 端		TT：	TT：等

有关文件名的规定：

文件，一般是指存放在软盘上的一批信息，为了表明它，寻找它和使用它，就要给取个名字，其一般格式为：设备名：文件名·文件类型。

在PDP-11/03系统中，此处的设备名只能为DY0（或SY），或者为DY1（或DK）。它指出后面给出的文件存放在0*软盘上，或在1*软盘上。

文件名，通常由英文字母开头的字符（由A..Z及0..9中的字符）组成。在PDP-11计算机系统中，文件名的字符数一般不允许超过六个，它由系统给出，或由用户自己定义。

文件类型，又叫文件扩展名，一般由三位字符组成，它有特定的含义。如：

PAS，代表该文件是PASCAL源程序；

MAC，代表MACRO（宏汇编语言）源程序；

LST，为列表文件；

OBJ，为二进制浮动地址文件；

SAV，为内存象文件，即可直接装入内存加以执行的文件；

DAT，为数据文件；

COM，为间接命令文件；

BAK，为后备文件；

TEC，为文本编辑文件；

FOR，代表FORTRAN源程序；

SYS，为操作系统本身的文件；等等。

这些符号已由计算机系统确定，不能随意使用，是什么类型的文件，就用什么类型名来标志它。

设备名与文件名之间的冒号(:)，以及文件名与类型名之间的英文句号(.)，不能省略，也不能用其它字符（包括空格）代替。

还有两点请读者注意：一是在PDP-11系统中，可以把某些慢速设备看作为一个文件，如行打写成LP:，显示终端表示为TT:等。这时它们后面不能再写具体文件名，而且不能写成LP和TT，把冒号省掉是不行的。二是默认盘的问题。在一般情况下，要给出0*软盘上的一个文件的文件名应写上DY0:或SY:，在1*软盘上应写上DY1:，或DK:。为方便用户，凡是在1*软盘上的文件，DY1:，或DK:可以省略不写，只给出具体的文件就行了。这就是说，当文件名前未明确给出设备名时，就约定这个文件存放在1*软盘上。所以，1*软盘为默认(DEFAULT)设备，又称为工作盘或用户盘。用户的程序与数据一般应放在这个盘中。系统程序一般存放在0*盘中，故称0*盘为系统盘。当然如果有特殊需要，也不一定非这样用不可，这个问题我们不讨论了。

下面，看几个正确的和错误的文件名的例子：

DY1:A.PAS 这是一个存放在1*盘上名字为A的PASCAL源程序。

DK:A.PAS 与DY1:A.PAS相同。

A.PAS 与DK:A.PAS相同。

DY0:PASCAL.SAV 存放在0*盘上的PASCAL编译程序，也可以写作：

SY:PASCAL.SAV。但是必须注意，DY0:（或SY:）不能省略。

DY1:A.MAC 这是一个存放在1*软盘上名字为A的宏汇编文件，也可以写成DK:A.MAC 或 A.MAC。

PASCAL.OBJ PASCAL语言所用的库程序，是二进制浮动程序。它是由系统提供的，这种写法，表明它存放在用户盘上。如果要用存放在系统盘上的这一库程序，则应写成SY:PASCAL.OBJ或DY0:PASCAL.OBJ。

A.COM	存放在用户盘中的名字为A的间接命令文件，即它的内容都是系统命令或键盘命令。
TT:	指显示终端，可以作为输入设备，也可作为输出设备用。注意，此时TT: 的后面不应再跟具体的文件名。
LP:	指行式打印机，只能做输出设备用。此时，LP:后面不能再写具体文件名。
上面的都是正确的文件名。下面举几个错误的文件名。	
DY2:A.PAS	PDP 机软盘只有DY0, DY1, 没有DY2, 所以表示错误。
ABCDEFGH.MAC	文件名超过六个字符，不合法。
A.PAS1	文件类型超过三个字符。
SY:A*B.SAV	文件名A*B中有非法字符*。
DK A.PAS	DK与A之间少了冒号(:)。
DY0:A B.OBJ	文件名中有空格字符。
DK: A.PAS	文件名之前不应有空格字符A, 有了空格字符则不合法。
TT	因缺乏冒号(:), 成为不合法的文件名。
LP	不合法的文件名，因LP后面缺少冒号(:)。

§1. PASCAL源程序的建立与修改

这里所讲的建立与修改 PASCAL 源程序，是指使用系统中给出的某种文本编辑程序 (EDITOR)，通过键盘把用户编好的PASCAL 源程序送进计算机里，说得更准确些是把它作为一个文件送到软盘里。如果发现它有编辑方面的错误，如字符打错，格式不合语法要求等，可以在键盘输入的过程中，或在其它任何时间将该文件调出，用文本编辑的方式加以修改。

在RT-11的V3版本中，有两个文本编辑程序。一个是EDIT，即通常说的编辑程序，另一个是TECO (TEXT EDIT AND CORRECT)。本书只介绍TECO的使用，因为它使用更为方便。有关EDIT的使用方法，可以参阅有关资料。

TECO是一种功能很强的编辑程序，可以用来编辑任何格式的ASCII文本，如各种语言的源程序，书籍，资料，稿件，书信等等。下面详细地介绍TECO的使用方法。

TECO有两种工作方式，即命令方式与立即方式，立即方式比命令方式更好用，主要是更直观，使用简便。所以本节主要介绍立即方式。

为了使用立即方式，系统中必须有一个VT-52或VT-100的显示终端，而且在软盘中有下列文件：

```
SY:TECO.SAY
SY:INIT.TEC
SY:LOCAL.TEC
SY:VT52.TEC
SY:VT100.TEC
DK:PASCAL.OBJ
```

DK : T.COM

一、进入TECO的立即方式 当终端处在等待键盘命令状态（又称为监控命令状态），即在显示终端屏幕上显示提示符“·”时，为了进入TECO的立即方式，可打入键盘命令：

.EDIT/EXEC -SY:INIT.TEC

在按下回车键之后，在屏幕最上端将显示出：

File name: ××××××.××× (第一种形式)

或 File name: (第二种形式)

其中××××××.×××代表某文件名及其类型，是在这之前在用户盘最后编辑或显示过的文件，如A.PAS，或USER01.PAS，等等。

如果用户盘从未编辑或显示过文件；或者用户盘编辑文件后曾初始化；或者上一次显示时也是“File name:”，即在冒号之后未打入文件名。在上面这三种情况下，即出现第二种形式。

如果要建立一个新的文件，在第二种形式的时候，可直接打入要建立的新文件的文件名。若是第一种形式时，即 File name: 后面有一个老文件的文件名时，则首先按 CTRL/U（即同时按 CTRL 键和 U 键），删除机器给出来的（老的文件的）文件名。或者用 DELETE 键，一个字符一个字符地删除之。然后用键盘打入要新建的文件名及其类型，例如 USER02.PAS，再按回车键。这时显示终端的屏幕上的 File name: USER02.PAS 立即消失，并在左上角显示立即方式的符号“■”，表示TECO已作好准备，可以接受输入。这时就可以用键盘打入要新建的文件内容。每打入一个字符，“■”这个立即方式的标记号往后移一个格，即它始终处在打入的文件的最后位置上。整个文件输入完毕，应打回车键，使“■”处在文件最末的单独一行上。否则，编译程序不处理最后打进去的那一行内容。

二、结束输入，退出立即方式 当文件输入已经完成，必须打 CTRL/C，即同时按 CTRL 键和 C 字符码键。这时TECO转为命令方式，在终端屏幕最下一行显示提示符“*”。此时，有两种选择：

1. 把刚打入的文件存贮在用户盘里，使这个新建的文件可供随时调用。同时要使系统回到等待键盘命令状态，即监控命令状态。这时应该打入键盘命令串 E X ESC ESC，此时终端显示 *EX\$ \$。

ESC 键，是终端上的一个控制键，它没有对应可显示（或打印）的字符，按它后系统将回送一个“\$”符号代替。

在系统存好文件之后，又进入等待键盘命令状态，在 *EX\$ \$ 下面显示“·”。

2. 把刚打入的内容全部作废（或者刚才再显的文件内容已经存入盘中，没有必要再存），由立即方式直接退到键盘命令方式，应打入字符串 CTRL/C ESC ESC。

CTRL/C 也是控制键，按它后系统回送符号 ^C。此时，屏幕上显示 *^C\$ \$，系统立刻回到等待命令状态。

请注意，在打了第一个 CTRL/C 键，终端屏幕左下方显示“*”之后，如果需要再回到立即方式，继续输入或编辑文件时，可以打如下键盘命令串：MI ESC ESC，此时在终端上显示 *MI\$ \$。系统马上回到立即方式，即可继续输入文件或进行程序修改。

图10.1-1给出几个命令后状态的相互关系。

要说明一点，在建立与修改文件的过程中，要反复地调入TECO。如果每次都要打 EDIT/EXEC -SY:INIT.TEC 就显得很麻烦。为此，专门建立一个间接命令文件 T.COM，其

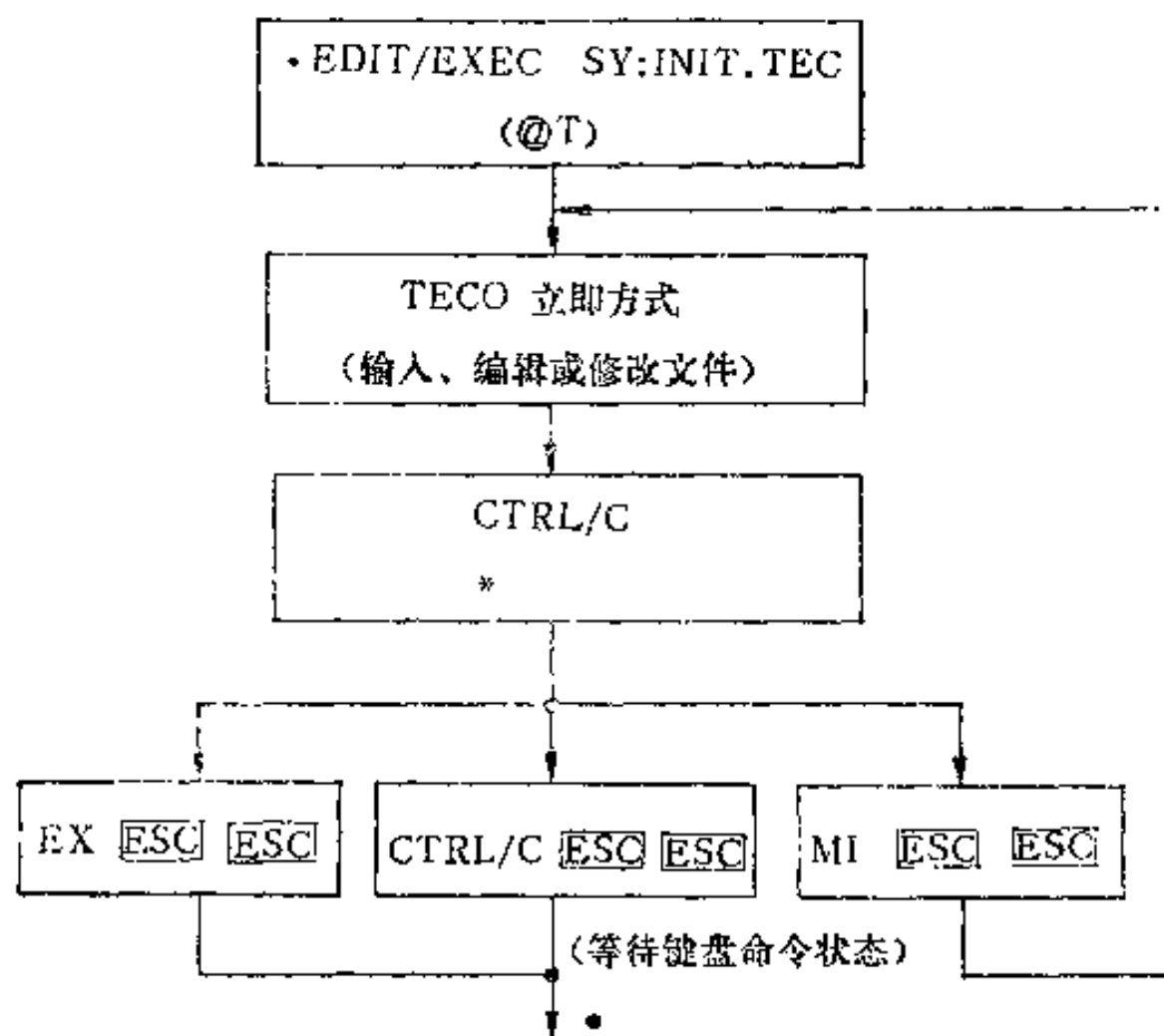


图10.1-1 几个键盘命令和系统状态的关系

内容就是EDIT/EXEC SY:INIT.TEC。此后，就可以把T.COM作为一条命令使用。RT-11操作系统规定，要运行一个间接命令文件中的内容，应首先打@键，紧接着打间接命令文件的名称，再打回车键（↵代表按下回车键）就行了。所以每当要打EDIT/EXEC SY:INIT.TEC命令使用TECO时，只要打@T↵就行了。在RT-11操作系统管理下的PDP-11计算机系统，人们总是用@T↵进入TECO的立即方式。

三、文件的修改与校正 文件在打入过程中，或打入后运行过程中，或进一步提高程序质量重新审查文件时，经常会发现这样那样的错误，可以用TECO进行修改与校正。其中在TECO的立即方式下校正更为直观和方便。其步骤大致如下：首先调出相应文件，并进入TECO的立即方式。实现的办法是打入键盘命令@T↵，之后屏幕上显示File name: ××××××。如果显示的文件名，正是要校正的程序文件，则可按回车键。如果显示的文件名，不是需要校正的文件，则删除这一文件名，换成所需要的文件名，然后再回车。这时终端上显示的就是要校正的文件的内容。在这时即可对文件进行修改和编辑。通常情况下，终端显示的总是文件的开头部分，而且在头一个字符下有一个闪烁的“—”符号，人们称它为光标（又叫指针）。它表示现在可以对它附近的内容进行删除，插入等操作，以校正程序。如果其它地方要修改，则应将光标移到那儿，然后进行相应的修改。“■”这个符号总停留在文件的最后，表示这一文件的结束。

下面介绍移动光标和进行修改的具体操作方法。请注意，这些操作必须在TECO的立即方式下才能进行。

1. 移动光标（即指针）位置的方法

如果在按↑↓←→诸键前，先按ESC键和数字n（n为正整数），则相应的动作进行n次。如ESC24↓，则指针下移24行。

例：如果文件的程序如下，指针在K处（即在字母K下面）。

(1) 按字按行移动的方法

所按键盘字符	操作结果说明
↑	指针上移一行
↓	指针下移一行
←	指针左移一个字符
→	指针右移一个字符
BACK SPACE	指针移到所在行的行尾
③ (辅助键盘)	指针移到所在行的行首
① (辅助键盘)	指针移到文件的开头
② (辅助键盘)	指针移到文件的结尾
④ (辅助键盘)	指针移到上一行的行首
⑤ (辅助键盘)	指针移到下一行的行首

A	B	C	D	E	F	1	2	3	4	5	6	7	8	9	0	G	H	
I	J	<u>K</u>	L	M	N	O	4		2		6		7		5	P	Q	R
S	T	U	V	W	X	Y	Z											
■																		

按字按行移动指针举例说明

键盘命令	指针位置	说明
↑	C	上移一行, 指针由K处移至C处
↓	U	指针下移一行, 由K处移至U处
←	J	左移一字符, 指针由K处移至J处
→	L	右移一字符, 指针由K处移至L处
BACK SPACE	R	指针移至本行行尾, 即R处
③ (辅助键盘)	I	指针移至本行行首, 即I处
② (辅助键盘)	■	指针移至文件程序底部末尾处
① (辅助键盘)	A	指针移至文件程序顶部开头处

续表

键 盘 命 令	指 针 位 置	说 明
③ (辅助键盘)	S	指针下移一行至行首, 即S处
④ (辅助键盘)	A	指针上移一行至行首, 即A处
ESC 3→	N	指针右移三个字符, 至N处
ESC 15→	P	指针右移十五个字符, 至P处

(2) 按给定的字符串移动的方法

用搜索命令键, 把指针移至指定的字符串所在位置。这是比按字按行移动的方法更快更方便的另一种办法。

使用方法与步骤如下:

第一, 按辅助键盘的检索键[ENTER]键, 或者[]键, 按下后在显示终端顶部一行显示: SEARCH:

第二, 在SEARCH: 显示后, 用键盘打入需要检索的字符串, 比如PQR。

第三, 按辅助键盘的[.]键, 每按一次, 指针从当前位置下移到第一次遇到的字符串PQR的后边。如果指针以后的程序里没有这一字符串, 指针将回到文件的开头位置。

2. 插入和删除的操作

(1) 插入操作

对PDP-11系列的计算机系统, 插入是很方便的。把指针移到要插入的位置, 顺序按下应插入的字符键串, 刚打入的字符插在指针的前面位置, 指针上面的那个字符和它后面的全部字符都顺序后移。

(2) 删除操作

删除操作命令及其说明如下

所 按 键 盘 字 符	操 作 结 果 说 明
DELETE	删除指针前面的一个字符
⑤ (辅助键盘)	删除指针所在位置上的一个字符
CTRL/K	删除从指针开始到该行行尾的全部字符 (包括回车与换行控制字符)
CTRL/D	删除从指针开始到该行行尾的全部字符, 但保留该行中的回车与换行控制字符。
CTRL/U	删除指针所在行中从行首到指针之间的全部字符。
⑥ (辅助键盘)	删除最后打进去的那一个字符

注: 上述各命令 (CTRL/U命令除外) 前打 [ESC] n, (n为正整数), 则该命令重复执行n次。

删除操作命令应用例题，假设文件如下，指针在K处

A	B	C	D	E	F	1	2	3	4	5	6	7	8	9	0		G	H	
I	J	<u>K</u>	L	M	N	O	4		2		6		7		5		P	Q	R
S	T	U	V	W	X	Y	Z												
5																			

删除操作举例说明

键 盘 命 令	功 能 说 明
DELETE	删除字母J，字符KLMN……PQR往前移一字符位置，指针仍在K处。
⑤ (辅助键盘)	删除字符K，字符LMN……PQR都顺序往前移一字符位置，指针在L处。
CTRL/K	删除KLMN……PQR这些字符，下一行的字符STUVWXYZ上移，接在J的后面，指针在S处。
CTRL/D	将KLMN……PQR这些字符全部删掉，但保留该行的回车与换行控制字符，下一行字符留在原处不动，指针仍在原处。
CTRL/U	删除字符IJ，KLMN……PQR全部字符顺序前移2个字符位置，K和指针在行首。
[ESC] [2] DELETE	删除字符K前面2个字符，KLMN……PQR前移2个字符位置，指针在行首的K处。
[ESC] [3] ⑤	删除字符KLM，字符NO4……PQR顺序前移3个字符位置，指针在N处。

3. 移动文本中信息块的操作

如果在建立和修改文件的过程中，发现有一段程序的位置错了，怎么办？这当然可以把这一段程序全部删除，然后再在正确的位置上重新打入。可是这要花费机时。能否用几个简单的操作，把这部分内容输到正确的位置上去？这就是要介绍的移动文本中的信息块的操作。这种操作常用于把文件中的一部分内容（这一部分程序，在整个文件中需多次重复出现，而只需作少量修改）在适当的位置重复若干次，以节省打字时间。这可以借助于辅助键盘上的PF1键和PF3键实现。具体操作如下：

首先，把指针移到要移动的信息块的第一行行首，然后打[ESC]键，并接着打进要移动的信息块的行数n，再打PF1键。这时，要移动的信息块内容已被记忆下来。

第二步，把指针移到需要的位置，即信息块将要移动到的那个位置的下一行的行首，再打下PF3键。这时，被移动的信息块被正确地插入了，终端屏幕上显示这一结果。

请看下面的例题。设文件某一段程序如下：

VAR

I:1..M; J:1..N; K:1..P;

A:ARRAY(1..M, 1..P) OF INTEGER;

F:TEXT;

如果把光标(指针)移到第三行行首，按[ESC] [2] PF1，然后把光标再移回到第三行行首。

再按PF3键，则上面一段程序变成下面的模样。即把第三行第四行内容复制一遍，插在原来第三行之上。

```
VAR
    I: 1..M; J: 1..N; K: 1..P;
    A: ARRAY(1..M,1..P) OF INTEGER;
    F: TEXT;
    A: ARRAY(1..M, 1..P) OF INTEGER;
    F: TEXT;
```

4. 开辟空行

如果要在文本的某个位置插入一行内容，或在某一行语句后空出若干行，可用开辟空行命令，其命令格式如下：

首先把指针移到需要开行的位置的行首，按辅助键盘的数字键⑦。这样就在指针所在行的上方开出一空行（按一下空一行，按n次则开出n个空行）。

例：文件某一段程序如下所示

```
BEGIN
    READ(N1);
    RESET(F1, N1);
    :
```

如果在BEGIN下面一行要插入“WRITELN(‘PLEASE INPUT N1’);”要开一空行。把指针移到第二行行首，按辅助键盘数字键⑦，则在第二行上方开出一空行，如下面：

```
BEGIN
    READ(N1);
    RESET(F1, N1);
    :
```

有了空行，便可插入（打入）语句。

5. 扩展方式——PF2键的功能

在使用TECO的立即方式的工作过程中，有这样的情况：临时转入命令方式工作一下，然后马上转回立即方式继续工作，这就是所谓扩展方式。它是借助于PF2键实现的。具体操作如下：在立即方式进行文本编辑过程中，在某处需要转入命令方式，此时按下辅助键盘的PF2键，之后在终端屏幕上的最上一行将显示：

Command,

这时即可按命令方式进行操作，打入需要的命令。如果命令执行成功，计算机系统将自动回到立即方式。如果命令失误，或者需要从命令方式回到立即方式，可打入键盘命令串：MI[ESC][ESC]，系统马上退回立即方式。

PDP-11/03系统，在TECO立即方式不能打右花括弧“}”，如果要打该符号，可以用下述命令：按下PF2键，出现“Command,”，在其后打入键盘命令串1251[ESC][ESC]。这样在按下PF2键前指针的位置，打出一个右花括弧“}”。

四、使用TECO编辑文件时的注意事项 使用TECO的立即方式进行文本编辑，有几点注意事项。

1. 要调用TECO并用它的立即方式, 首先(在键盘命令状态)打入@T \swarrow , 随后在屏幕上显示“File name:”或File name:×××××. ×××”。如果在“File name:”之后没有文件名, 或显示的不是需要的文件名, 那末需要重新打入新的文件名及其类型, 例如“FIL07B.PAS”, 然后打回车键。如果是新建一个文件, 则即可以打入文件内容; 如果是修改、编辑已有的文件, 则在屏幕上显示该文件的内容后, 进行相应的修改与编辑。

2. 在完成打入文件内容, 或完成修改与编辑之后, 一定要先打入CTRL/C, 之后在屏幕上显示符号“*”。紧接着打入键盘命令串EX[ESC] [ESC]。随后在屏幕上显示*EX\$, 表示文件已送到用户盘中。否则打入的文件(或修改编辑的文本)没有送到盘中, 造成前功尽弃。在编辑很长的文件时, 尤其要注意这一点, 可以打入一部分内容, 存入(盘中)一部分, 再重新调出来, 继续打入。

3. 当指针已移到行首, 如按下删除键([DELETE]键), 则在行首处出现奇异符号“f”, 如再接着按动其它键后, 符号“f”消失。但此时已潜伏着问题, 造成难以查找的错误。在这种情况下, 每次移动指针, 这一行字符总会闪动, 有人称之为“闹鬼”。这样的程序如不改正, 在编译时总是通不过, 而且不指出具体错误。修改的办法是将闪动的这一行全部删除, 重新打入。如果在文本编辑时出现了符号“f”, 即指针在行首时按一次[DELETE]键, 那末只要马上再按一次[DELETE]键, 错误即可消除。

4. 文件必须用回车键结束。就是说文件最后一行打入后, 必须按下回车键, 使光标和方块在最末一行的行首出现。否则PASCAL编译程序不处理这个文件的最后一行。

5. 指针在行首, 同时它又是文本的末尾时, 不能用右移命令, 只能用空格键右移指针。

§2 PASCAL 程序的运行步骤和方法

一、PASCAL 源程序运行的步骤 如果一个PASCAL源程序已完成编辑, 并存放在用户盘上。在PDP-11/03计算机系统上进行(确切地说应是OMSI PASCAL-1语言编写的)源程序, 分下列四个步骤完成:

第一步, 对PASCAL源程序进行编译, 产生一个相应的宏汇编程序和一个编译列表文件;

第二步, 对得到的宏汇编程序进行汇编, 产生一个相应的二进制浮动程序和一个汇编列表文件;

第三步, 把得到的二进制浮动程序与PASCAL库程序进行链接, 得到一个可以直接装入内存加以运行的目标程序;

第四步, 把上述目标程序装入内存并加以运行, 产生一定的结果。

上述步骤的次序不能变更, 后一步要用前一步的结果。如果前一步未进行, 或执行过程中失误, 得不到正确的相应结果, 则下一步无法进行。另外, OMSI PASCAL-1编译程序生成的是宏汇编代码, 而不是机器代码。这与大多数的PASCAL编译版本不同, 所以编译后还要再进行汇编, 方能得到机器代码。另外一些编译程序版本, 可以直接产生机器代码, 那时就用不到执行第二步的汇编操作。

二、几个键盘命令的使用

1. 可直接装入内存加以运行的SAV类型文件的运行。

而前已经指出, 只有SAV类型的文件才能够直接装入内存并加以运行。这类文件可以

是系统程序（一般存放在系统盘中），也可以是用户程序（通常存放在用户盘中）。

运行系统盘中的 SAV 类型文件，用键盘命令 R，如：

```
R PASCAL.SAV
```

意思是把存放在系统盘中的 PASCAL 编译程序装入内存并启动运行它。

```
R MACRO.SAV
```

则表示把保存在系统盘中的宏汇编程序装入内存并启动运行它。

运行用户盘中的 SAV 类型的文件，用键盘命令 RUN，如：

```
RUN A.SAV
```

表示把保存在用户盘中的名字为 A 的 SAV 类型文件装入内存并启动运行它。

由于只有 SAV 类型文件方能用 R 或 RUN 命令去运行它，所以在使用键盘命令 R 或 RUN 命令时，只需要给出运行文件的名称即可，SAV 类型可以省略。也就是说，下面所示

R PASCAL 与 R PASCAL.SAV 效果完全一样；

RUN A.SAV 与 RUN A 效果完全相同。

还有，计算机系统能分辨用 R 或 RUN 命令运行的是系统盘或是用户盘中的文件。所以，在键盘命令 R（或 RUN）之后，运行的文件名之前不必再给出设备名（SY: 或 DK:）。

2. 在行式打印机上输出字符文件

这时要用键盘命令 PRINT（可简化为 PRI 或 P）。在命令后输入要打印的文件名，在 PRINT 和文件名之间至少要有个空格。如果要打印的是几个文件（最多允许六个），在文件名之间用一个逗号隔开。如：

```
PRINT SY:INIT.TEC, DK:A.PAS, SMP.DAT, DB01.MAC
```

即可在行打机上输出系统盘的 INIT.TEC 文件，用户盘上的 A.PAS, SMP.DAT 和 DB01.MAC 文件。

要说明的是，能够在行式打印机打印的只能是字符文件，如 PAS 类型，MAC 类型，LST 类型和 DAT 类型，而二进制文件，如 OBJ 类型，SAV 类型，是不能直接在行打机上打印的，因为它们不是由字符组成的。

3. 在终端屏幕上显示字符文件

在终端屏幕上显示字符文件的一种方法是用键盘命令 TYPE，使用方法与 PRINT 命令类似。在命令 TYPE（可简化为 TYP）之后隔一空格，打入要显示的文件名称及其类型。如：

```
TYPE ABC.LST
```

表示在终端上显示名字叫 ABC 的列表文件。

前面已经提到，用 @T 可以在终端屏幕上显示各种文件，而且是经常采用的一种方法。因为人们显示文件的目的，往往是为修改程序，这就要求在 TECO 的立即方式下进行。用 .@T 显示文件即可满足这一要求。而用 TYPE 命令只是显示一次文件而已，不能复看，也不能修改，但用此命令一次可逐个显示一批（比如说五个以下）文件。

4. 文件的复制、换名与删除

键盘命令 COPY（可简化为 COP）、RENAME（可简化为 REN）、DELETE（可简化为 DEL）分别是完成文件复制、换名与删除的命令。如：

```
COP A,PAS B,PAS
```


它表示把用户盘上 A.PAS 这个文件，复制一份内容格式完全相同的文件，取名为 B.PAS，存放在同一盘上。

COP SY:PASCAL.OBJ DK:PASCAL.OBJ↵

它表示把系统盘上的 PASCAL.OBJ 文件复制到用户盘上，文件名和类型不动。在这种情况下，“DK:”后的 PASCAL.OBJ 可以省略。如果文件名要更改，则自然要打入新的文件名。

COP PF05A.DAT+PF05B.DAT PF05.DAT↵

它表示将两个文件的内容相加，复制出一份新的文件。把若干个同属字符类型的文件合并为一个文件，在很多情况下是很有用处的。

REN A.PAS B.PAS↵

它表示把用户盘中原来叫 A.PAS 的文件，换名为 B.PAS。

REN SY:A.PAS SY:B.PAS↵

它表示把系统盘上原来称 A.PAS 的文件，换名为 B.PAS。B.PAS 前的“SY:”不能省略，因为换名这个命令，只是更改文件的名称，文件本身并不改变，依然存放在原来的盘中。

DEL C.PAS↵

它表示要删除用户盘上名叫 C.PAS 这个文件。由于删除是个“危险”的操作，计算机系统为了防止用户粗心大意，把有用的文件误删除，因此系统会在上面删除操作命令串后，显示：

DEL DK:C.PAS?

如果确认要删去用户盘中的 C.PAS 这个文件，那末可打入字母 Y（表示 YES）。如果发现原删除操作有误，不应删除 C.PAS 这个文件，那末打入字母 N（表示 NO，不删。实际上打入除 Y 以外的任何一个键即可。）。

如果要删除一串相同类型的文件，例如说要把用户盘上的所有后备（BAK 类型）文件删除，可打如下命令：

DEL *.BAK↵

计算机系统会自动显示用户盘上全部 BAK 类型的文件，并且逐个询问用户是否要删除，马上要求用户回答，打入 Y 表示删除，打入字母 N（Y 以外任何一个键即可）表示不删除。如：

.DEL DK:SMP01.BAK? Y↵

.DEL DK:SMP02.BAK? Y↵

.
.
.

.DEL DK:FIL11.BAK? Y↵

它表示把用户盘中全部 BAK 类型文件删除，如果其中一部分有用，则在显示相应文件名后打入字母 N，凡是这样处理的文件依然保存在用户盘中，可继续使用。

如果要删除这样一批文件：它们的名称的头几个字符相同，其文件类型各不相同。例如本书中的第四章的程序全部以 CTL 三个字母开头命名的，在运行过程中会产生各种类型的

文件。如果要删除其中一部分或全部文件，可以打入如下键盘命令串：

```
DEL CTL*. * ✓
```

计算机系统会把用户盘中的凡是 CTL 开头的全部文件逐个自动显示出来，让用户来选择要删除还是要保留。具体操作同上。

如果是毫无疑问地删除某些文件，为节省时间，可把 DEL 命令改为 DEL/NOQ。这样，计算机系统就不再显示询问的内容，直接删除就是了。

在使用复制、换名和删除命令时请注意以下几点：第一，不要轻易改变文件的类型，如 COP A.PAS B.MAC 或 REN A.PAS A.OBJ 都是错误的；

第二，复制和换名不一样。复制的结果使盘中有了两个仅名称不同、内容完全相同的文件，换名命令执行后，原来那个名称已不复存在；

第三，删除命令的使用要慎重。对那些确实无用的文件要及时删除，对有用的文件不要误删，尤其不要轻易删除系统盘中的文件，以免损害整个系统的正常运行。

三、PASCAL 源程序的编译 对 PASCAL 源程序进行编译，是由保存在系统盘中的(文件名为 PASCAL.SAV) PASCAL 编译程序完成的。

1. PASCAL 源程序的编译和 MACRO 源程序文件、列表文件的产生

把编译程序 PASCAL.SAV 装入内存并启动运行，用的键盘命令是：

```
.R PASCAL ✓
```

在这之后，编译程序立刻输出字符 *，位置为：

```
* 
```

空格行方框是人为加的，提示用户在这个位置打入回答的内容：编译程序编译产生的两个文件 (MAC 文件和 LST 文件) 的文件名和被编译的源程序的程序名。正确回答后，编译程序就对给出的源程序进行编译，并产生相应的结果。

回答的内容一般为：

```
FILE1.MAC, FILE2.LST=FILE3.PAS
```

```
或 FILE1.MAC=FILE3.PAS
```

```
或, FILE2.LST=FILE3.PAS
```

这里的 FILE3 为被编译的源程序的文件名，其类型只能是 PAS 类型。FILE1 为编译产生出来的宏汇编语言源程序的文件名，其类型只能是 MAC 类型。FILE2 为编译产生的列表文件，只能是 LST 类型。

在详细介绍源程序编译之前，提醒读者注意以下几问题：第一，由于 FILE1, FILE2, FILE3 这三个文件的类型都是固定的类型，所以在回答时将其类型省略，只写文件名，编译程序能自行处理；第二，对每一个源程序，编译程序可以产生两个文件，如果这两个文件用户均需要，则在等号左边给出两个文件名。也可以只让它产生一个文件 (MAC 类型)，这时在等号左边只给出一个文件名即可。但如果需要第二个文件 (LST 类型)，则一定要在它的文件名之前加一个逗号；第三，如果被编译的一个源程序是由几个 (最多允许六个) 文件组成的，则在等号右边依次给出每个文件名，相邻文件名之间用逗号分隔开；第四，文件名 FILE1, FILE2 可以是 LP: 或 TT:，即用行打或显示终端代替。

2. 编译命令和编译选择开关的功能

在对源程序进行编译的时候，用户经常要求对编译程序的某些功能进行控制。如上面介绍过的几种编译命令中对如何生成列表文件的控制。这些控制是通过在编译命令行中给出的

编译选择开关来实现的。

几种编译命令的功能简介

编译命令	功能及执行结果
• R PASCAL ✓ • A = A ✓	用PASCAL编译程序，把等号右边的PASCAL源程序A.PAS编译成MAC类型程序A.MAC。用户盘中必须有源程序类型的名称为A的源程序，其它类型编译程序不予承认。
• R PASCAL ✓ • A, A = A ✓	用PASCAL编译程序，把等号右边的PASCAL源程序A.PAS编译成MAC类型程序A.MAC和列表文件A.LST。
• R PASCAL ✓ • , LP := A ✓	用PASCAL编译程序，把等号右边的PASCAL源程序A.PAS编译成列表文件，并在行式打印机上输出。
• R PASCAL ✓ • , LP := A/N / (或 • , LP : /N = A ✓)	基本功能同上，所不同的是在行式打印机上输出列表文件时，只列出那些有错误的语句。
• R PASCAL ✓ • , TT := A ✓	用PASCAL编译程序，把等号右边的PASCAL源程序A.PAS编译成列表文件，并在终端屏幕上显示。
• R PASCAL / • , TT : /N = A ✓	功能基本同上，所不同的是在终端屏幕上显示的只是些有错误的错句。

下面把 OMSI PASCAL-1 所使用的这些选择开关及其作用编成下表的形式

编译选择开关一览表

开关	作用
/D	编译输出中包含调试信息。这些调试信息包括变量名称，过程和语句的位置等，由链接程序把这些信息自动地加到程序中去，这在PASCAL调试程序中能得到很好应用。 /D开关常常和/S开关一起使用，这样可使调试程序中的信息与相应的源程序语句位置发生联系，在这种情况下，产生一个列表文件是必要的，它可使PASCAL调试程序确定源程序行在文本中的位置。
/E	形成一个外部过程的定义。程序中确定的全程一级的过程和函数可以作为外部过程或函数进行编译。其它单独编译的PASCAL程序也能够调用这些外部过程或函数。这些过程和函数的名称的头六个字符用作作为全程符号，做为相应子程序的入口。
/F	此开关对没有硬件浮点部件的机器（如FIS或FPP）用来加快实数操作的速度。选择这开关使编译程序产生子程序的调用，而不是解释FIS指令，结果是提高了一点速度，但程序要大一些。
/L	允许列表文件列出源程序。 这开关和/N开关配合使用，用来控制PASCAL编译所建立的列表文件和内容。
/N	不允许列出正确的源程序语句，当这开关生效时，仅仅把包含有错误的那些语句写在列表文件里。
/S	编译输出MAC类型文件注有源程序。用这个开关可以清楚地看到每个PASCAL语句所对应的宏汇编指令段。 这个开关常和/D开关一起使用，作用已在/D开关栏中说明。

这些编译选择开关放在编译命令中，表示开关在整个编译过程都起作用。至于开关位置可以写在等号的右边，也可以写在等号左边，两者效果相同。

比如：

.R PASCAL 和 .R PASCAL
* A, A = A/S * A, A/S = A

两个编译命令都表示编译所生成的宏汇编文件都注有相应的源程序。

为了说明编译命令和编译选择开关的功能，分别举例如下。

例10-1 今有源程序 RUN01.PAS，经过编译命令：.R PASCAL✓

* LP:/S=RUN01✓ 输出宏汇编

文件 RUN01.MAC

其源程序，编译命令和输出结果如 FIG RUN01 PROGRAM 所示。

```
PROGRAM RUN01;
VAR
  I, J:INTEGER;
BEGIN
  I := 1;
  I := 2;
  J := 3;
  J := I + J
END.

.R PASCAL
• RUN01/S=RUN01 (OR • LP:/S=RUN01)
  (OR • TT:/S=RUN01)

OUTPUT:
(•      RUN01.MAC
, PROGRAM RUN01;
, VAR
      .GLOBL $RESR6, $RESR5, $KORE, $SAV10, $VER
      .GLOBL $BEGIN, FILE, $TTY, $END
      .RADIX 10
, I, J:INTEGER;
, BEGIN
      $VER=59.
$BEGIN:
      ADD      #4, $KORE
      JSR      %7, $B127
      .GLOBL $B127
, I := 1;
      MOV      #1, @%5
, I := 2;
      MOV      #2, @%5
, J := 3;
      MOV      #3, 2(5)
, J := I + J
, END.
      MOV      @%5, %0
      ADD      2(5), %0
```

```

MOV    %0, 2(5)
JMP     $END
.END

```

FIG RUN01 PROGRAM

3. 编译命令和编译选择开关使用举例

在 PASCAL 源程序中，也可以引用编译选择开关中的任意一个，用来控制编译生成的 MAC 类型文件。在编译过程中如果遇到注释符以后的第一个字符是 \$，则编译程序会自动寻找是哪一个编译开关。例如：

(* \$ S

表示编译输出文件注释有相应的源程序，在其后加符号 “+” 或 “-”，分别表示所用开关的状态是开或关，也就是编译开关是有效还是无效。(* \$ S + *) 表示以下编译输出的 MAC 类型文件注释有相应的源程序。(* \$ S - *) 则表示以下编译输出的 MAC 类型文件不注释相应的源程序。

例10-2 有 PASCAL 源程序 RUN02.PAS，在源程序中用编译开关 /S，即在第二个语句上注有 (* \$ S - *)，在第三个语句上注有 (* \$ S + *)。这样在编译输出的 MAC 类型文件中，没有注出相应的第二个源程序语句 I := 2；而在第二个语句后又注释了下面的源程序语句。其源程序、编译命令和输出结果见 FIG RUN02 PROGRAM。

```

PROGRAM RUN02,
VAR
  I, J:INTEGER,
BEGIN
  I:=1;
  ( * $ S - * )
  I:=2;
  ( * $ S + * )
  J:=3;
  J:=I+J
END.

R PASCAL
* RUN02/S=RUN02 (OR * TT:/S=RUN02)
                (OR * LP:/S=RUN02)

OUTPUT:
( *          RUN02.MAC
, PROGRAM RUN02,
, VAR
    .GLOBL $RESR8, $RESR5 $KORE, $SAV10, $VER
    .GLOBL $BEGIN, FILE, $TTY, $END
    .RADIX 10
, I, J:INTEGER,
, BEGIN
    $VER=59.
$BEGIN:

```

```

        ADD      #4, $KORE
        JSR      %7, $B127
        .GLOBL   $B127
,      I:=1;
        MOV      #1, @%5
,      (* $S-*)
        MOV      #2, @%5
,      J:=3;
        MOV      #3, 2(5)
,      J:=I+J
,      END.

        MOV      @%5, %0
        ADD      2(5), %0
        MOV      %0, 2(5)
        JMP      $END
        .END

```

FIG RUN02 PROGRAM

一个程序编写以后，在计算机上调试程序是必不可少的步骤。但对于一个很大的源程序来说，并不是一次编写成的，往往是在积累过去许多小程序运行通过的基础上，分若干次逐步调试运行的。因此，调试一个大程序时，并不需要对整个程序一次进行。为减少调试程序所占内存单元的数量，往往只对那些新编写的或者容易出问题的地方用调试程序。实现的办法是合理地使用编译选择开关/D，即在需要调试的源程序段前后分别加(*\$D+*)和(*\$D-*)。

编译选择开关/L是允许列表文件列出源程序。当编译命令不加其它开关时，/L开关的作用是默认的。也就是说没有使用/L开关，仍然认为允许列表文件列出源程序。

例10-3 对PASCAL源程序RUN3A.PAS，进行如下编译命令：

```

.R PASCAL✓
*, RUN3A=RUN3A✓

```

则输出列表文件中列出全部源程序，并在错误语句下指出错误的地点和性质。见FIG RUN03A PROGRAM。

```

PROGRAM RUN03A,
VAR
  I, J:INTEGER,
BEGIN
  I:=1,
  I:=2,
  J=3,
  J:=I+J
END.

```

(• 以下对上述 RUN03A PAS 进行编译并输出 •)
R PASCAL
• RUN03A=RUN03A (OR •, LP:=RUN03A)

(OR • ,TT:=RUN03A)

OUTPUT:

```
(•          RUN03A.LST          •)
LINE STMT LEVEL NEST SOURCE STATEMENT
  1                PROGRAM RUN03A,
  2                VAR
  3                I, J:INTEGER,
  4                BEGIN
  5      1      1      1      I:=1;
  6      2      1      1      I:=2;
  7      3      1      1      J=3;
      •••••      BAD EXPRESSION
  8      4      1      1      J:=I+J
  9                END.
```

ERRORS DETECTED: 1

FREE MEMORY: 14170 WORDS

FIG RUN03A PROGRAM

编译选择开关/N，不允许在列表文件中列出正确的语句，只准列出包含有语法错误的语句。比如对 RUN03B.PAS，在编译命令中加上了/N开关，结果在输出的列表文件中，只列出有错误的语句。其源程序、编译命令和输出结果如 FIG RUN03B PROGRAM 所示。

```
PROGRAM RUN03B,
VAR
  I,J:INTEGER,
BEGIN
  I:=1;
  I:=2;
  J=3;
  J:=I+J
END.
```

.R PASCAL

• ,RUN03B/N=RUN03B (OR • ,LP:/N=RUN03B)
(OR • ,TT:/N=RUN03B)

OUTPUT:

```
(•          RUN03B.LST          •)
LINE STMT LEVEL NEST SOURCE STATEMENT
  7      3      1      1      J=3;
      •••••      BAD EXPRESSION
ERRORS DETECTED: 1
FREE MEMORY: 14049 WORDS
FIG RUN03B PROGRAM
```

请注意，由于列表文件不进入 TECO，因此用 @T 命令进入 TECO 立即方式，找不到列表文件。要寻找列表文件可用 TYPE 和 PRINT 命令。

编译选择开关 /L 和 /N 配合使用，可以让用户决定列表文件的内容。

例10-4 现有 PASCAL 源程序 RUN04.PAS，希望第一个语句要列出源程序，以下只允许列出有错误的语句。实现的办法，源程序，编译命令和选择开关的使用，以及输出结果在 FIG RUN04 PROGRAM 上所示。

```

PROGRAM RUN04;
VAR
  I,J:INTEGER;
BEGIN
  ( * $ L + * )
  I:=1;
  ( * $ L - * )
  I:=2;
  J=3;
  J:=1+J
END

.R PASCAL
* ,RUN04/N=RUN04 (OR * ,LP:/N=RUN04)
                (OR * ,TT:/N=RUN04)
.OUTPUT,
( *                RUN04.LST                * )
LINE STMT LEVEL NEST SOURCE STATEMENT
   5                                ( * $ L + * )
   6      1      1      1      I:=1;
   9      3      1      1      J=3;
                                ( * $ L - * )

* * * * *      BAD EXPRESSION
ERRORS DETECTED: ;
FREE MEMORY: 14050 WORDS
      FIG RUN04 PROGRAM

```

此外，还有两个编译开关 A 和 T，它们只能以 (* \$ A - *) 和 (* \$ T - *) 的形式，或 (* \$ A + *) 和 (* \$ T + *) 的形式用在 PASCAL 源程序中，以便指明 PASCAL 编译程序是否应产生对数组下标范围与堆栈范围进行检查的代码。正常情况下，编译程序应产生对数组下标范围与堆栈范围进行检查的代码，这对保证程序的正确运行是十分必要的。只有当人们经过反复运行，确信一个常用的程序已正确无误，肯定不再存在数组下标越界与堆栈溢出错误时，才能在 PASCAL 源程序中使用 (* \$ A - *) 与 (* \$ T - *)，来解除对数组下标范围与堆栈范围进行的检查，以便使目标程序小一些，运行速度快一点。

源程序 RUN05.PAS 有两处数组越界，由于在第一处数组越界前使用了开关 (* \$ A - *)，所以此处的数组越界错误不予检查，程序继续运行。而在第二处由于恢复了数组越界检查，所以程序运行到这儿指出数组越界，并停止执行。程序及开关 (* \$ A - *) 的使用

见 FIG RUN05 PROGRAM 所示。

```
PROGRAM RUN05;
VAR
  A:ARRAY (1..8) OF INTEGER;
  I:INTEGER;
BEGIN
  (* $A- *)
  FOR I:=1 TO 10 DO A[I] :=I;
  WRITELN('DO NOT CHECK ARRAY BOUNDS. ');
  WRITELN('      PROGRAM RUNNING GO ON ');
  (* $A+ *)
  FOR I:=1 TO 10 DO A[I] :=I;
  WRITELN('ARRAY BOUNDS ; PROGRAM STOP ');
END.
```

OUTPUT:

```
DO NOT CHECK ARRAY BOUNDS,
      PROGRAM RUNNING GO ON ;
Array bounds error - from PC 003170
FIG RUN05  PROGRAM
```

4. PASCAL 源程序和宏汇编语言源程序的衔接

在 PASCAL 源程序中，可以用特定的注释方式插入宏汇编语言源程序段。注释符（* \$ C 表示以下是宏汇编语言源程序段，而注释符 * ）表示插入部分结束。

宏汇编语言源程序段的程序，可以访问 PASCAL 源程序中的变量。如果源程序中全程变量是 A，局部变量是 B，则在宏汇编部分访问变量 A、B 分别用 A（%5）和 B（%6）表示。

在 FIG RUN06 PROGRAM 这个程序中，主程序用 PASCAL 赋值语句给全程变量 A 赋予初值 1，然后用宏汇编 MOV 指令改变其内容，使 A 所在单元的值 12345。过程 PR01 中局部变量 B 也被进行类似的处理。请看 FIG RUN06 PROGRAM 所示的程序和输出的结果。

```
PROGRAM RUN06;
VAR
  A:INTEGER;
PROCEDURE PR01;
VAR
  B:INTEGER;
BEGIN
  B:=2;
  WRITELN('ORIGINAL VALUE: B=' .B:2);
  (* $C
    MOV #22, B(%6)
  *)
```



```

      WRITELN('RESULT OF MOV #22,B(%6):B= ', B:2);
      WRITELN
    END;
  BEGIN  (•   MAIN   •)
    A:=1; PR01;
    WRITELN('ORIGINAL VALUE:A= ',A:2);
    (• $C
      MOV #12345, A(%5)
      •)
    WRITELN('RESULT OF MOV #12345, A(%5):A= ', A:5)
  END.
OUTPUT:
ORIGINAL VALUE:B=2
RESULT OF MOV #22, B(%6):B=22
ORIGINAL VALUE:A=1
RESULT OF MOV #12345, A(%5):A=12345
      FIG RUN06   PROGRAM

```

四 对 MAC 类型程序的汇编 对编译源程序得到的 MAC 类型程序进行汇编的键盘命令是:

```
.R MACRO✓
```

该命令的功能是把系统盘里可运行的 MACRO.SAV 文件调到内存; 终端显示 *, 等待汇编命令:

```
* A=A✓
```

汇编命令把等号右边的 MAC 类型程序 A.MAC 汇编成目的程序 A.OBJ。文件类型 MAC 和 OBJ 是计算机系统默认的, 所以省略不写, 但必须有 MAC 类型的名称为 A 的文件 A.MAC。

系统可以对多个 MAC 类型程序进行逐个汇编。如果汇编结束, 即没有汇编程序进行汇编了, 则在键盘上敲入 CTRL/C, 随后在终端上显示

```
* ^C
```

计算机系统立即返回到键盘命令状态, 终端上显示 “.”。这表示汇编已经完成, 在用户盘上已经有了对应 MAC 类型程序 (比如 A.MAC) 的目的程序 (如 A.OBJ)。

在一般情况下, 如果对源程序编译没有错误, 那末汇编一般也没有错误。而一旦编译有错误, 那汇编就无法进行下去。

汇编也能生成汇编列表文件, 但此时的汇编命令应为:

```
.R MACRO✓
```

```
* A, A=A✓
```

但是这样的汇编命令产生的汇编列表文件会把同名的编译列表文件冲掉。如果两种列表文件都要保留, 那末在汇编命令中应该对汇编列表文件改名, 如:

```
.R PASCAL✓
```

```
* A, A=A✓
```

```
* ^C
```

.R MACRO ✓

* A, B = A ✓

此时得到的 B.LST 就是汇编列表文件, 而 A.LST 则是编译列表文件。

OBJ 类型的文件, 比如上面编译汇编得到的 A.OBJ, 是浮动的二进制的程序, 机器不能直接运行。另外 OBJ 类型程序也不能用键盘命令 TYPE 在终端屏幕上显示。如果特殊需要显示二进制的程序, 如 A.OBJ, 则需打入如下键盘命令:

.DUMP/TER A.OBJ ✓

为了得到能直接装入内存加以执行的程序, 必须对汇编得到的目的程序进行链接操作。这是下面的内容。

五、对二进制浮动程序的链接过程 对汇编得到的二进制浮动程序和 PASCAL 的库程序进行链接的键盘命令为

.R LINK ✓

表示把系统盘里的可运行的链接程序 LINK.SAV 调到内存, 终端屏幕显示 *, 等待链接命令:

* A=A, PASCAL ✓ 或 * A=A, SY:PASCAL

表示要将等号右边的 A.OBJ 程序 (PASCAL 源程序经过编译和汇编产生的目的程序) 和 PASCAL 的库程序 (PASCAL.OBJ) 进行链接, 产生一个名称为 A 的能直接装入内存并能加以执行的 SAV 类型程序 A.SAV, 由于 SAV 类型和 OBJ 类型是默认的, 所以在链接命令中省略不写。但要注意, 在等号右边进行链接的程序均为 OBJ 类型文件, 并且 PASCAL.OBJ 必须复制到用户盘上, 或在系统盘上有此文件。在一行上一次链接的 OBJ 类型文件一般不允许超过六个, 超过六个时, 可以利用开关 /C, 在下一行继续给出, 如果还有, 可以在第三行继续给出, 这样链接的结果产生一个 SAV 类型文件。此时, 在终端屏幕上显示符号 *, 等待新的链接命令。如果已没有新的文件需要进行链接, 则可打入 CTRL/C 键, 此时屏幕上显示:

* ^C

表示链接过程结束, 计算机系统回到键盘命令状态。链接过程产生了 A.SAV 程序。

六、目标程序的运行 PASCAL 源程序经过编译、汇编和链接过程以后, 得到了能直接装入内存并能加以执行的 SAV 类型文件, 如 A.SAV 程序。要运行这个程序, 只要进行如下键盘命令:

.RUN A ✓

如果源程序成功地编译、汇编和链接, 那末就有可能得到预期的输出结果。至于因为程序设计中的逻辑错误等原因, 得不到输出结果, 或者输出不是预期的结果, 这个问题在下面一节中进行详细讨论。

七、无用文件的删除 PASCAL 源程序在运行过程中生成的中间文件 MAC 类型和 OBJ 类型文件, 如 A.MAC, A.OBJ, 往往对用户并没有用处, 应该予以删除, 为此用如下键盘命令

.DEL/NOQ A. (MAC, OBJ) ✓

删除文件时, 加上开关 /NOQ, 是告诉计算机系统无需再向用户询问, 直接加以删除。A.MAC 和 A.OBJ 等程序是无用的中间文件。及时删除无用文件是非常重要的, 否则不仅让

无用文件白白地占据软盘存储空间，而且经常会发生因文件太多，而使需要正常运行的文件不能编辑和运行。

八、程序运行的技巧 上面讲到的 PASCAL 源程序运行的五步操作(或称五个过程)，是在程序员自己控制下一步一步地完成的。为了更简便起见，用户可以把用到的五条命令写成一个标准命令文件：如通常采用的 A.COM，然后通过执行 A，即在键盘命令中执行 .@A↵，便可间接地运行 A.COM 中的所有命令。

标准命令文件 A.COM 如下：

R PASCAL

A, TT:=A

R \MACRO

A=A

^C

R LINK

A=A, PASCAL

^C

RUN A

DEL/NOQ A. (MAC, OBJ)

请注意，A.COM 文件中的 ^C，是通过主键盘敲入符号 ^ 键和 C 键形成的，不是敲入 CTRL/C 所构成。

如果有了标准命令文件 A.COM，源程序是 A.PAS，那末运行这个程序，在键盘命令状态只要执行：

.@A↵

计算机系统会自动地一步一步去运行 A.PAS 这个源程序，这就是所谓程序运行的技巧。

如果源程序的名称不是 A，只要在运行这个源程序前将它的名称改为 A 就行了。但通常的办法是将源程序(比如 SMP05.PAS)复制一次，将其名称改为 A。实现的办法是：

.COPY SMP05.PAS A.PAS↵

.@A↵

因为源程序文件往往有许多，其名称不是随便乱题，往往考虑到分类的方便和应有的物理意义，有一定的规律性。例如本书第三章简单的程序设计中的源程序用 SMP (simple) 作名称开头，按其顺序编为 SMP01, SMP02, ……。第九章文件的例题，其程序名称的头三个字符用 FIL (file)，几个源程序的名称定为 FIL01, FIL02, ……。这些源程序的编制调试都要花出辛勤的劳动，因此一般都要保存好。只是在运行这些源程序时，临时复制为 A.PAS，这是程序运行的技巧而已，并非必须这样做。

如果对应的源程序(比如 SMP05.PAS)的目标程序要经常运行，则在 .@A↵ 运行完毕后，执行：

.COPY A.SAV SMP05.SAV↵

以后运行 SMP05.PAS 这个源程序，只要执行：

.RUN SMP05↵

这样可以省去源程序的编译、汇编和链接的过程，使程序运行的时间缩短。另一方面，

由于 SAV 类型文件是二进制表示的, 它所占软磁盘的盘区数要比源程序大得多, 为了节省软盘存储的空间, 以保存更多有用的源程序, 对于不是经常运行的目标程序常常删去。

九、带有外部过程的程序的运行 在第五章第五节中, 已经比较全面地介绍了外部过程和外部函数的特点、使用的理由和运行的方法。在这一部分内容中, 主要是简单地介绍包含外部过程 (包括外部函数, 下同) 的程序的组成, 运行的注意事项和具体步骤, 以及对包含有四个以上外部过程时如何解决链接的问题。

包含有外部过程的程序, 至少有三个以上的独立的文件: 全程变量文件, 主程序文件和外部过程文件。外部过程的文件可以是一个, 也可以是很多个, 它们独立存放在盘中, 可供多个主程序或其它外部过程所调用。

外部过程文件建立的方法, 与建立标准 PASCAL 源程序相同, 其名称的前六个字母数字字符, 应该是程序能识别的 (或称单值的)。外部过程的文件中, 往往在源程序开始之前加上外部过程开关

(* \$ E + *) 或 { \$ E + }

这样做的好处是容易提醒人们来识别这是外部过程文件。当然也不一定这样做。但在这种情况下编译外部过程文件时, 在编译命令中必须加上外部过程开关 /E, 否则外部过程文件无法正确编译。

全程变量文件服务于主程序文件和外部过程文件。它必须含有包括在主程序及其调用的外部过程中全部适用的全程变量, 它可以包含程序首部, 包括全程的标号定义、类型定义、内部过程和内部函数的说明等。在编译主程序、编译主程序所调用的每个外部过程时 都要用到它。全程变量文件必须单独作为一个文件存放在盘中。

主程序文件, 应该说它是整个程序的执行部分。在主程序文件中不能再写程序首部、全程变量等内容。在主程序执行部分之前, 对它所调用的每个外部过程, 必须予以说明, 格式如下:

```
PROCEDURE 过程名1(形式参数表); EXTERNAL;  
PROCEDURE 过程名2(形式参数表); EXTERNAL;  
.....  
PROCEDURE 过程名 n(形式参数表); EXTERNAL;
```

这儿的进程名应该是外部进程名, 但必须加上 EXTERNAL 说明后, 才说明要调用的是外部过程文件。形式参数表可有可无, 视具体情况而定。

运行带有外部过程的程序时, 有几点特别要加以说明:

1. 在编译时, 必须把主程序文件、各个被调用的外部过程文件, 分别与全程变量文件一一进行编译, 产生各自的 MAC 类型程序文件;
2. 对编译产生的 MAC 类型程序文件, 必须分别进行汇编, 产生对应的 OBJ 类型程序文件;
3. 在链接过程时, 必须将上一步产生的各个 OBJ 类型程序文件与 PASCAL 库程序 PASCAL.OBJ 链接在一起。如果外部过程超过四个, 也就是在一行上一次链接的 OBJ 类型文件超过六个时, 可利用开关 /C, 在下一行继续给出。也可以对外部过程经编译、汇编产生的相应的 OBJ 类型程序文件进行复盖处理, 以减轻对内存的要求。链接的结果得到一个 SAV 类型的文件。
4. 运行结束, 删除无用文件时, 必须将其主程序文件和各个外部过程文件在编译、汇编过程中产生的 MAC 类型程序文件和 OBJ 类型程序文件一一删去。

下面，我们以一个包含三个外部过程文件的程序的运用为例子，作一说明。
全程变量文件 GLOVAR.PAS 如下：

```
( * GLOVAR.PAS * )  
VAR  
    I, J, K: INTEGER;
```

主程序文件 MAIN.PAS 如下：

```
( * MAIN.PAS * )  
PROCEDURE EXAA, EXTERNAL;  
PROCEDURE EXBB, EXTERNAL;  
PROCEDURE EXCC, EXTERNAL;  
BEGIN  
    EXAA;  
    EXBB;  
    EXCC;  
    I := 1; J := 10 * I; K := 100 * I + 10 * J;  
    WRITELN('I = ', I:2, ' J = ', J:2, ' K = ', K:3);  
    WRITELN('END OF MAIN PROGRAM')  
END.
```

外部过程 1 文件 EXAA.PAS 如下：

```
( * EXAA.PAS * )  
( * $E+ * )  
PROCEDURE EXAA;  
BEGIN  
    WRITELN('END OF EXAA')  
END;
```

外部过程 2 文件 EXBB.PAS 如下：

```
( * EXBB.PAS * )  
( * $E+ * )  
PROCEDURE EXBB;  
BEGIN  
    WRITELN('END OF EXBB')  
END;
```

外部过程 3 文件 EXCC.PAS 如下：

```
( * EXCC.PAS * )  
( * $E+ * )  
PROCEDURE EXCC;  
BEGIN  
    WRITELN('END OF EXCC')  
END;
```

用下述键盘命令，即可一步一步地运行上述这个程序，

```

.R PASCAL
• MAIN=GLOVAR, MAIN
.R PASCAL
• EXAA=GLOVAR, EXAA
.R PASCAL
• EXBB=GLOVAR, EXBB
.R PASCAL
• EXCC=GLOVAR, EXCC
.R MACRO
• MAIN=MAIN
• EXAA=EXAA
• EXBB=EXBB
• EXCC=EXCC
• ^C          (• CTRL/C •)
.R LINK
• PROG=MAIN, EXAA, EXBB, EXCC, PASCAL
• ^C          (• CTRL/C •)
.RUN PROG
.DEL/NOQ EXAA.(MAC,OBJ)
.DEL/NOQ EXBB.(MAC,OBJ)
.DEL/NOQ EXCC.(MAC,OBJ)
.DEL/NOQ MAIN.(MAC,OBJ)

```

为了检查主程序文件和每个外部过程文件的编译有无错误，在编译命令中应使用有关编译选择开关，在终端屏幕上显示列表文件清单，以指出编译时文件错误的地点、性质和个数。这样便于修改程序。例如要检查外部过程文件 EXBB.PAS 有无编译错误，可用下述键盘命令：

```

.R PASCAL
• EXBB, TT:=GLOVAR, EXBB

```

上述用人工输入键盘命令的办法来运行包含外部过程的办法，是经常用到的，读者应该逐步掌握。也可以仿照在前面讲到的建立标准命令文件 A.COM，而运行 @A 的办法，将上述步骤建立一个工作命令文件，比如 C.COM，内容如下：

```

(• C.COM •)
R PASCAL
MAIN=GLOVAR, MAIN
R PASCAL
EXAA=GLOVAR, EXAA
R PASCAL
EXBB=GLOVAR, EXBB
R PASCAL
EXCC=GLOVAR, EXCC

```

```

R MACRO
MAIN=MAIN
EXAA=EXAA
EXBB =EXBB
EXCC =EXCC
^C          (* NOT CTRL/C *)
R LINK
PROG=MAIN, EXAA, EXBB, EXCC, PASCAL
^C          (* NOT CTRL/C *)
RUN PROG
DEL/NOQ EXAA.(MAC,OBJ)
DEL/NOQ EXBB.(MAC,OBJ)
DEL/NOQ EXCC.(MAC,OBJ)
DEL/NOQ MAIN.(MAC,OBJ)

```

如果要运行上述程序（包含全程变量文件 GLOVAR.PAS，主程序文件 MAIN.PAS 和三个外部过程文件 EXAA.PAS, EXBB.PAS, EXCC.PAS），只要执行.@C/即可。

如果上述程序中的三个外部过程文件中，没有加外部过程标记开关（* \$E+*），如下所示：

```

          (* EXAA.PAS *)
PROCEDURE EXAA,
BEGIN
  WRITELN ('END OF EXAA')
END;

          (* EXBB.PAS *)
PROCEDURE EXBB,
BEGIN
  WRITELN('END OF EXBB')
END;

          (* EXCC.PAS *)
PROCEDURE EXCC,
BEGIN
  WRITELN('END OF EXCC')
END;

```

那末，在外部过程文件编译时，必须用编译选择开关/E，编译命令如下

```

.R PASCAL
* MAIN=GLOVAR, MAIN
.R PASCAL
* EXAA=GLOVAR EXAA/E
.R PASCAL
* EXBB=GLOVAR, EXBB/E

```

```
.R PASCAL
* EXCC=GLOVAR, EXCC/E
```

其它汇编、链接、运行和删除命令同上。

这个程序运行的结果如下：

```
OUTPUT:
END OF EXAA
END OF EXBB
END OF EXCC
I= 1, J=10, K=200
END OF MAIN PROGRAM
```

下面，简单介绍链接过程中的复盖处理。在上面的例题中，仅包含三个外部过程。如果运行包含四个以上的外部过程的源程序时，对主程序和每个外部过程经编译、汇编而得到的相应的 OBJ 类型文件，进行复盖链接是最方便的如：

```
.R LINK
* MAIN=MAIN, PASCAL/C
* EXAA/O:1/C
* EXBB /O:1/C
* EXCC /O:1/C
  :
* EXFF/O:1
```

在外部过程文件数在四个以下时也可以使用复盖链接法。复盖链接法在链接很大的程序而内存容量又有限制时，是节省内存单元的一个有力措施。

运行一个包含有六个外部过程文件的源程序的实例，其源程序清单，运行步骤的键盘命令，运行技巧建立一个标准工作命令文件，运行结果等，如 FIG RUN07 PROGRAM 所示。

```
( * GLOVAR.PAS * )
PROGRAM RUN07 (INPUT,OUTPUT),
VAR
  I, J, K :INTEGER;
( * ..... * )
( * MAIN.PAS * )
PROCEDURE EXAA; EXTERNAL;
PROCEDURE EXBB; EXTERNAL;
PROCEDURE EXCC; EXTERNAL;
PROCEDURE EXDD; EXTERNAL;
PROCEDURE EXEE; EXTERNAL;
PROCEDURE EXFF; EXTERNAL;
BEGIN
  EXAA;
  EXBB;
  EXCC;
  EXDD;
```



```

EXEE,
EXFF,
I:=1; J:=10*I; K:=100*I+10*J;
WRITELN('I=',I:2,', J=',J:2,', K=',K:3);
WRITELN('END OF MAIN PROGRAM')
END.
( ..... )
      ( •   EXAA.PAS   • )
( • $E+ • )
PROCEDURE EXAA;
BEGIN
  WRITELN('END OF EXAA')
END;
( ..... )
      ( •   EXBB.PAS   • )
( • $E+ • )
PROCEDURE EXBB;
BEGIN
  WRITELN('END OF EXBB')
END;
( ..... )
      ( •   EXCC.PAS   • )
( • $E+ • )
PROCEDURE EXCC;
BEGIN
  WRITELN('END OF EXCC')
END;
( ..... )
      ( •   EXDD.PAS   • )
( • $E+ • )
PROCEDURE EXDD;
BEGIN
  WRITELN('END OF EXDD')
END;
( ..... )
      ( •   EXEE.PAS   • )
( • $E+ • )
PROCEDURE EXCEE;
BEGIN
  WRITELN('END OF EXEE')
END;
( ..... )
      ( •   EXFF.PAS   • )
( • $E+ • )

```

```

PROCEDURE EXFF,
BEGIN
  WRITELN('END OF EXFF')
END,
( • ..... • )
  .R PASCAL
  • MAIN=GLOVAR,MAIN
  .R PASCAL
  • EXAA=GLOVAR,EXAA
  .R PASCAL
  • EXBB=GLOVAR,EXBB
  .R PASCAL
  • EXCC=GLOVAR,EXCC
  .R PASCAL
  • EXDD=GLOVAR,EXDD
  .R PASCAL
  • EXEE=GLOVAR,EXEE
  .R PASCAL
  • EXFF=GLOVAR,EXFF
  .R MACRO
  • MAIN=MAIN
  • EXAA=EXAA
  • EXBB=EXBB
  • EXCC=EXCC
  • EXDD=EXDD
  • EXEE=EXEE
  • EXFF=EXFF
  • ^C
  .R LINK
  • PROG=MAIN,PASCAL/C
  • EXAA/O:1/C
  • EXBB/O:1/C
  • EXCC/O:1/C
  • EXDD/O:1/C
  • EXEE/O:1/C
  • EXFF/O:1
  • ^C
  .RUN PROG
  .DEL/NOQ EXAA.(MAC,OBJ)
  .DEL/NOQ EXBB.(MAC,OBJ)
  .DEL/NOQ EXCC.(MAC,OBJ)
  .DEL/NOQ EXDD.(MAC,OBJ)
  .DEL/NOQ EXEE.(MAC,OBJ)
  .DEL/NOQ EXFF.(MAC,OBJ)

```

```

    DEL/NOQ MAIN (MAC,OBJ)
(•   PROG.COM   •)
R  PASCAL
MAIN=GLOVAR,MAIN
R  PASCAL
EXAA=GLOVAR,EXAA
R  PASCAL
EXBB=GLOVAR,EXBB
R  PASCAL
EXCC=GLOVAR,EXCC
R  PASCAL
EXDD=GLOVAR,EXDD
R  PASCAL
EXEE=GLOVAR,EXEE
R  PASCAL
EXFF=GLOVAR,EXFF
R  MACRO
MAIN=MAIN
EXAA=EXAA
EXBB=EXBB
EXCC=EXCC
EXDD=EXDD
EXEE=EXEE
EXFF=EXFF
AC
R  LINK
PROG=MAIN,PASCAL/C
EXAA/O:1/C
EXBB/O:1/C
EXCC/O:1/C
EXDD/O:1/C
EXEE/O:1/C
EXFF/O:1
AC
RUN PROG
DEL/NOQ EXAA.(MAC,OBJ)
DEL/NOQ EXBB (MAC,OBJ)
DEL/NOQ EXCC.(MAC,OBJ)
DEL/NOQ EXDD.(MAC,OBJ)
DEL/NOQ EXEE.(MAC,OBJ)
DEL/NOQ EXFF.(MAC,OBJ)
DEL/NOQ MAIN.(MAC,OBJ)
OUTPUT:
END OF EXAA

```

```

END OF EXBB
END OF EXCC
END OF EXDD
END OF EXEE
END OF EXFF
I= 1, J=10, K=200
END OF MAIN PROGRAM
FIG RUN07 PROGRAM

```

上面的例题，每个外部过程的执行部分仅仅为输出一段字符串，没有进行其它运算，没有其它复杂功能，因此显得十分单调。之所以这样做，主要是考虑举例要简单明了。输出一段字符串，表示这一外部过程已被调用并正确地执行。例题中六个外部过程一律加上外部过程标记开关（* \$ E+ *），所以在外部过程编译时，一律不用编译选择开关/E。实际情况可不是千篇一律。读者应该根据自己的具体要求灵活地加以运用。

§3 查找并改正 PASCAL 源程序中的错误

在 PASCAL 程序设计过程中，往往出现两类错误。一类是语法错误，就是在程序中说明与使用数据、标识符及各种语句等过程中出现的某些违犯 PASCAL 语法规则的错误。另一类是非语法错误，通常是指逻辑错误。就是说程序在编译时没有任何错误，甚至汇编、链接时都正常，但在运行得到的目标程序时，却得不到结果，或者得到的不是预期的结果。这表明程序设计者在程序中未能正确地表达自己的意图，或者计算方法有某种程度的错误，等等。

查找与改正前一类错误，主要借助于编译过程中生成的列表文件；查找后一类错误，往往借助于 PASCAL 的辅助调试程序 DEBUGGER。下面分别介绍这两部分的内容。

一、PASCAL 源程序中的语法错误的检查与改正

1. 列表文件

前面已经讲到，查找程序中的语法错误，主要借助于编译过程中生成的列表文件。在上一节中介绍过，编译选择开关/L 是允许列表文件列出源程序。执行相应的编译命令，即可输出列表文件列出全部源程序。现有 PASCAL 源程序 DEB01.PAS，由 OMSI PASCAL-1 编译程序所生成的列表文件，如 FIG DEB01 PROGRAM 所示。

```

LINE  STMT LEVEL  NEST  SOURCE STATEMENT
1          PROGRAM DEB01 (INPUT, OUTPUT);
2          { DEB01.PAS.....DEBUGGER EXAMPLE 1 }
3          TYPE
4          ARR=ARRAY (1..10) OF INTEGER;
5          VAR
6          I, J:INTEGER; A:REAL; CH:CHAR; AR:ARR;
7
8          PROCEDURE ADD (VAR I2: INTEGER);
9          VAR

```

10				J2, K2:INTEGER; AR2:ARR;	
11					
12				FUNCTION SUB (I3:INTEGER):INTEGER;	
13				VAR	
14				J3:INTEGER;	
15				BEGIN (• START OF SUB •)	
16	1	3	1	J3:=100;	{ SUB, 1 }
17	2	3	1	SUB:=J3-I3;	{ SUB, 2 }
18	3	3	1	WRITELN('SUB=',(J3-J3):3)	{ SUB, 3 }
19				END; (• END OF SUB •)	
20					
21				BEGIN (• START OF ADD •)	
22	1	2	1	K2:=2;	{ ADD, 1 }
23	2	2	1	K2:=222;	{ ADD, 2 }
24	3	2	1	FOR J2:=1 TO 4 DO	{ ADD, 3 }
25	4	2	2	AR2 (J2) :=2 * J2;	{ ADD, 4 }
26	5	2	1	I2:=AR2 (1) +SUB (10);	{ ADD, 5 }
27	6	2	1	WRITELN ('I2=', I2:3)	{ ADD, 6 }
28				END;	
29				(• END OF ADD •)	
30					
31				BEGIN (• START OF MAIN •)	
32	1	1	1	A:=3.1415;	{ MAIN, 1 }
33	2	1	1	A:=7.345;	{ MAIN, 2 }
34	3	1	1	CH:='A';	{ MAIN, 3 }
35	4	1	1	ADD (1);	{ MAIN, 4 }
36	5	1	1	WRITELN ('I=', I:3);	{ MAIN, 5 }
37	6	1	1	FOR J:=1 TO 10 DO	{ MAIN, 6 }
38	7	1	2	WRITELN (' J=', J:2);	{ MAIN, 7 }
39	8	1	1	WRITELN ('END.....')	{ MAIN, 8 }
40				END.	

FIG DEB 01 PROGRAM

列表文件清单分析如下:

第一列 (LINE), 标的是 PASCAL 源程序的行号, 每开始一行, 行号加 1。

第二列 (STMT), 标出语句在主程序或过程中的语句号。主程序, 或主程序调用的过程 (或函数), 或过程中的过程 (或函数), 它们开始的第一个语句为 1, 每增加一个语句, 语句号加 1。如果一行上写有二个以上语句, 那末标出的语句号是本行的第一个语句的语句号。在利用调试程序调试源程序时, 这些语句号作为找到或指出一个相应语句的标记。

第三列 (LEVEL), 标的是过程的层号。它指出一个过程或函数所处的位置。主程序的层号为 1, 主程序调用的过程 (或函数) 的层号为 2, 过程中定义的过程 (或函数) 的层号为 3 依此类推。在判定某些规则时, 层号是有用的。例如上述程序中 LEVEL 为 1 的主程序, 只能访问全程变量而不能访问过程中的局部变量。层号为 1 的主程序可以调用层号为

2 的过程而不能调用层号为 3 的函数。

第四列 (NEST)，表示程序中语句的嵌套数。在一种层号的开始，语句嵌套数为 1，在遇到结构型语句（例如一级套一级的复合语句、循环语句或重复语句等）时嵌套数加 1。NEST 的值表示该语句在程序分块结构中嵌套的深度。

列表文件的右侧标着 SOURCE STATEMENT，即列出 PASCAL 源程序。

2. 源程序中的语法错误的检查

如果 OMSI PASCAL-1 编译过程中发现源程序有错，通常先在出错语句的下一行的出错位置下面注脱字符‘^’，表示这里有错，并在下一行行首给出六个星号 * * * * *，接着给出指明错误类型或性质的有关信息。

编译过程中能检查出来的全部错误的信息，已被汇总在一起，详见附录六。

下面是一个有错误的 PASCAL 源程序的列表文件，如 FIG DEB 02 PROGRAM 所示。

```
LINE  STMT LEVEL NEST  SOURCE STATEMENT
1          PROGRAM DEB02 (INPUT, OUTPUT);
2          ( * DEB02.PAS.....DEBUGGER EXAMPLE 2 * )
3          TPE
      ^
* * * * * IMPROPER SYMBOL
4          ARR=ARRAY (1..10) OF INTEGER;
      ^
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
5          VAR
6          I, J:INTEGER, A:REAL, CH:CHAR, AR:ARR;
      ^      ^      ^      ^
* * * * * BAD TYPE SPECIFICATION
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
* * * * * BAD TYPE SPECIFICATION
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
7
8          PROCEDURE ADD (VAR I2:INTEGER);
      ^      ^
* * * * * BAD TYPE SPECIFICATION
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
9          VAR
10         J2, K2:INTEGER, AR2:ARR;
      ^      ^      ^      ^
* * * * * BAD TYPE SPECIFICATION
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
* * * * * BAD TYPE SPECIFICATION
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
11
12         FUNCTION SUB (I3:INTEGER):INTEGER;
      ^      ^
* * * * * BAD TYPE SPECIFICATION
```

```

* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
13          VAR
14          J3:INTEGER,
          ^      ^
* * * * * BAD TYPE SPECIFICATION
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
15          BEGIN      (• START OF SUB •)
          ^
* * * * * BAD PARAMETER
16          J3:=100;          { SUB, 1 }
          ^      ^
* * * * * MISSING') 'AT END OF PARAM LIST
* * * * * EXPECTED 'SEMI-COLON' MISSING. ASSUMED WHERE INDICATED.
* * * * * IMPROPER SYMBOL
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
17  1      2      1      SUB:=J3-J3;          { SUB, 2 }
          ^      ^
* * * * * MISSING BEGIN
* * * * * UNDEFINED OPERAND ..... ASSUMING INTEGER
18  2      2      1      WRITELN (' SUB= ', (J3-I3) :3)  { SUB, 3 }
          ^
* * * * * UNDEFINED OPERAND ..... ASSUMING INTEGER
19          END;          (• END OF SUB •)
20
21          BEGIN      (• START OF ADD •)
22  1      1      1      K2:=2;          { ADD, 1 }
          ^
* * * * * UNDEFINED SYMBOL
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
23  2      1      1      K2:=222;          { ADD, 2 }
          ^
* * * * * UNDEFINED SYMBOL
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
24  3      1      1      FOR J2:=1 TO 4 DO          { ADD, 3 }
          ^
* * * * * BAD FOR STATEMENT
25          AR2 (J2) :=2 * J2;          { ADD, 4 }
          ^
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
26  4      1      0      I2:=AR2 (1) +SUB (10);          { ADD, 5 }
          ^
* * * * * UNDEFINED SYMBOL
* * * * * ALL CHARACTERS IGNORED UNTIL SEMI-COLON
27  5      1      0      WRITELN ('I2= ', I2:3)          { ADD, 6 }
          ^
* * * * * UNDEFINED OPERAND ..... ASSUMING INTEGER
28          END;          (• END OF ADD •)
          ^
* * * * * MISSING '.' AT PROGRAM END

```

ERRORS DETECTED: 21

FREE MEMORY: 13960 WORDS

FIG DEB02 PROGRAM

检查源程序 A.PAS 的错误可用以下三种方法

(1) 执行如下编译命令和键盘命令:

```
.R PASCAL
```

```
* A, A=A
```

```
•TYPE A.LST
```

编译时, 同时生成 MAC 类型文件 A.MAC 和列表文件 A.LST。用 TYPE 命令在终端上显示列表文件的内容, 检查错误。

(2) 执行下述编译命令:

```
.R PASCAL
```

```
* A, TT:=A
```

编译时, 产生 MAC 类型程序 A.MAC 的同时, 在终端屏幕上显示列表文件 A.LST, 此时可观察错误。

(3) 执行如下编译命令

```
.R PASCAL
```

```
* A, LP:=A
```

编译时, 在生成 MAC 类型程序 A.MAC 的同时, 在行式打印机上输出列表文件, 可进行书面检查。上述三种方法, 只是检查错误。要修改源程序文件必须让系统进入 TECO 编辑方式。这三种方法中, 其中第二种应用最多。这种方法如果只是检查源程序的错误, 不生成宏汇编文件, 可按下列编译命令进行:

```
.R PASCAL
```

```
* , TT:=A
```

或者

```
.R PASCAL
```

```
* , TT:/N=A
```

表示只在终端上显示列表文件, 加了编译选择命令开关/N, 表示列表文件中只写那些有错误的语句。

3. 源程序错误的校正

通过编译命令输出列表文件的方法, 可以查出源程序的语法类错误。由于这种方法只是观察检查错误, 不能修改源程序, 所以要在观察时把错误的性质和地点记下来, 或者利用行式打印机输出错误清单。在系统进入 TECO 的立即方式时, 再修改源程序中的错误。

提请读者注意的是, 经常有这样的情况, 列表文件列出许多错误。是否按列表文件指出的那样去逐处修改呢? 否。往往是一个关键性错误会引起连锁反应, 造成一连串的错误。只要把这个关键性的错误找出以后, 可能问题一下子解决了。例如 DEB02.LST 这个列表文件, 指出存在二十一个错误, 但真正的错误只有一个, 类型定义 TYPE 误为 TPE。由于类型定义错了, 整个程序有关类型定义的地方均发生错误, 许多语句明明没有任何问题, 却标出了错误。

因此, 校正错误的原则为: 先程序首部和说明部分, 后执行部分和尾部; 先重点, 后一

般：先改正明显的，后校正隐蔽的。

还有，列表文件错误的标志脱字符‘^’，并不是指在真正错误之处，有时离错误的语句很远，而且标注的错误信息含意不清。这时，往往要在错误标志的前后寻找真正的错误所在。请看下而一个列表文件 DEB03.LST，如 FIG DEB03 PROGRAM 所示。

```

LINE  STMT LEVEL  NEST  SOURCE STATEMENT
1          PROGRAM DEB03 (INPUT, OUTPUT);
2          { DEB03.PAS.....DEBUGGER EXAMPLE 3 }
3      TYPE
4          ARR=ARRAY (1..10) OF INTEGER;
5      VAR
6          I, J:INTEGER; A:REAL; CH:CHAR; AR:ARR;
7
8      PROCEDURE ADD (VAR I2:INTEGER);
9      VAR
10         J2, K2:INTEGER; AR2:ARR;
11
12         FUNCTION SUB (I3:INTEGER) :INTEGER;
13         VAR
14             J3:INTEGER;
15         BEGIN          (* START OF SUB *)
16             I3:=100;          { SUB, 1 }
17             SUB:=J3-I3;       { SUB, 2 }
18             WRITELN (' SUB=, (J3-I3) :3) { SUB, 3 }
19         END;          (* END OF SUB *)
20
21         BEGIN          (* START OF ADD *)
22             K2:=2;           { ADD, 1 }
23             K2:=222;         { ADD, 2 }
24             FOR J2:=1 TO 4 DO { ADD, 3 }
25                 AR2 (J2) :=2 * J2; { ADD, 4 }
26                 I2:=AR2 (1) +SUB(10); { ADD, 5 }
27                 WRITELN ('I2=, I2:3) { ADD, 6 }
28
29             END;
30
31         BEGIN          (* START OF MAIN *)
32             1      2      1      A:=3.1415; { MAIN, 1 }
33             2      2      1      A:=7.345; { MAIN, 2 }
34             3      2      1      CH:='A'; { MAIN, 3 }
35             4      2      1      ADD (1); { MAIN, 4 }
36             5      2      1      WRITELN('I= ', I:3); { MAIN, 5 }
37             6      2      1      FOR J:=1 TO 10 DO { MAIN, 6 }

```

..... MISSING OPERATOR

```

38      7      2      1      WRITELN(' J=', J:2);          { MAIN, 7 }
39      8      2      2      WRITELN(' .....END.....')    { MAIN, 8 }
40                                     END.

..... EXPECTED 'SEMI-COLON' MISSING. ASSUMED WHERE INDICATED
..... IMPROPER SYMBOL
41
42      FIG DEB03      PROGRAM
..... FATAL ERROR: MISSING END
ERRORS DETECTED: 4
FREE MEMORY: 13803 WORDS
      FIG DEB03      PROGRAM

```

思考题：DEB03.LST 列表文件中，错误的真正原因是字符串 'SUB=' 误写为 'SUB=, 'l2:= ' 误为 'l2=。为什么错误标志指着 'l，从中有什么奥妙，试找出一点规律来。

另外，为了缩小寻找错误的范围，往往用注释符（* 和 *）或 { 和 } 将某些源程序套起来，这是为什么？请在实际中试用之。

二、源程序中逻辑错误的检查与改正 前面已经讲过，程序中的某些逻辑错误，往往要等到运行这一程序时才会暴露出来。所以又称这些错误为运行时的错误。要找出这些错误的位置和原因，一般是比较困难的。比较有效的办法，是能够追踪程序的运行过程，随时检查执行的是哪些语句，检查并改变某些变量的值，检查或追踪某些过程的调用情况等等。在多数计算机系统中，通过称作辅助调试程序的服务程序，如 EDP-11/03 计算机系统上的 ODT (On-line Debugging Technique)，给出了这种能力。我们这里不准备讨论它。我们在下面要介绍的，是在 OMSI PASCAL-1 的库程序 PASCAL.OBJ 中给出的一个叫做 DE-BUGGER 过程的调用与使用方法。不言而喻，它是专门为了调试 PASCAL 的目标程序而设计的，使用起来更方便。因为它能够把由它控制运行的目标程序与相应的 PASCAL 源程序对应起来。人们不妨称它为 PASCAL 程序的辅助调试程序。

下面将详细地介绍 DEBUGGER 这一辅助调试程序的使用方法，以及如何经过它查找程序中的某些逻辑错误。

1. PASCAL 辅助调试程序的调用

要调用 PASCAL 辅助调试程序，必须在编译源程序的编译命令行中提出来。原因有两个。首先，这一程序运行时要用到很多信息，如语句位置编号、变量名称及其对应的地址、语句内容等等，这些都要由编译程序来准备，也就是说必须告诉编译程序准备这些信息。其次，这一程序本身在 PASCAL.OBJ 中，编译程序必须安排好要调用它。具体做法是在编译程序的编译命令行中加选择开关 /S 与 /D。

如果要调试的源程序是 A.PAS，执行下述操作即可进入调试程序的入口，

```

.R PASCAL
* A, A=A/S/D
.R MACRO
* A=A
* ^C      (* CTRL/C *)
.R LINK

```

```

* A=A, PASCAL
* ^C          (* CTRL/C *)
.RUN A
PASCAL DEBUGGER V2.2
LISTING FILE NAME ? A
)

```

编译时使用了/D和/S两开关。/D开关使编译输出的MAC类型程序中,包括调试所需要的语句位置、变量名称的信息。/S开关可以使输出文件包含相应的源程序语句。编译时,同时生成一个列表文件A.LST。用户根据这个列表文件,才能知道各语句的位置。这样才能使用调试命令。

在执行如下命令后,终端将显示信息:

```

.RUN A
PASCAL DEBUGGER V2.2
LISTING FILE NAME ?

```

等待用户在键盘上打入编译该文件时生成的列表文件名。回答列表文件名A之后,终端显示调试命令提示符') ',表示已进入调试程序的入口,调试程序DEBUGGER已正常运行,等待用户打入调试命令。

如果回答的文件名不是编译该程序时生成的列表文件,则调试程序时,给出的语句就不一定对,甚至张冠李戴。

如果编译时不生成列表文件,则运行该程序时不询问列表文件名称,如下所示:

```

.R PASCAL
* A=A/S/D
.R MACRO
* A=A
* ^C          (* CTRL/C *)
.R LINK
* A=A, PASCAL
* ^C          (* CTRL/C *)
.RUN A
PASCAL DEBUGGER V2.2
)

```

表面上看,似乎调试程序运行正常,用户可以输入调试命令。但实际上,由于没有生成列表文件,无法判断各语句的位置,因此,不能设置断点,这样势必造成很难控制程序的运行情况。

2. 调试命令的应用

控制与调试一个程序的运行过程,是通过调试命令完成的。现将各调试命令汇总成表,见下页调试命令一览表。

下面以源程序A.PAS为例,使用DEBUGGER调试程序,对上述调试命令加以详细介绍。

源程序A.PAS如FIG DEB04 PROGRAM所示,

调 试 命 令 一 览 表

调 试 命 令	功 能 说 明
\$!	列出程序最后执行的十个语句
\$S	查看过程执行堆栈的内容
\$V	变量名称列表命令
proc, stmt, B 0; B 或 ; B	设置断点, proc为过程名, stmt为语句号 清除断点
; P n; P	从断点继续执行 如果设置断点, 程序通过断点n次; 如果设置单步则执行n个语句
; G n; G	开始执行程序 (与 ; P 功能相同) 功能同 n; P
1; T 0; T	置语句追踪方式 清除语句追踪方式
1; C 0; C	置过程追踪方式 清除过程追踪方式
1; S 0; S	置单步方式 清除单步方式
Var; W -1; W	监视变量值的变化 清除变量监视
Var/ Var/n	查看十进制整数变量 查看n个整数
Var\ Var\n	查看十进制字节变量 查看n个字节
Var% Var%n	查看实数变量 查看n个实数变量
Var' Var'n	查看字符变量 查看n个字符
Var=	打印变量地址
Var := number Var := 'text' Var := Rnumber	给变量赋以整数值 给变量赋以字符数据 给变量赋以实数值

运算符 '+' 和 '-' 是加, '-' 为减, '^' 是指针, 使用这些操作符可以查看复杂的记录或很复杂的数据结构。

```

PROGRAM A (INPUT, OUTPUT),
  { A.PAS.....DEBUGGER EXAMPLE 4 }
TYPE
  ARR=ARRAY (1..10) OF INTEGER,
VAR
  I, J:INTEGER; A:REAL; CH:CHAR; AR:ARR,
  PROCEDURE ADD (VAR I2:INTEGER);
  VAR
    I2, K2:INTEGER; AR2:ARR,

```

```

FUNCTION SUB (I3:INTEGER):INTEGER,
VAR
    J3:INTEGER,
BEGIN      (• START OF SUB •)
    J3:=100;                                { SUB, 1 }
    SUB:=J3-I3;                              { SUB, 2 }
    Writeln(' SUB=', (J3-I3):3)              { SUB, 3 }
END;      (• END OF SUB •)
BEGIN      (• START OF ADD •)
    K2:=2;                                    { ADD, 1 }
    K2:=222;                                  { ADD, 2 }
    FOR J2:=1 TO 4 DO                          { ADD, 3 }
        AR2 [J2] :=2 * J2;                     { ADD, 4 }
        I2:=AR2 [1] +SUB (10);                  { ADD, 5 }
        Writeln('I2= ', I2:3)                  { ADD, 6 }
    END;
        (• END OF ADD •)
BEGIN      (• START OF MAIN •)
    A:=3.1415;                                { MAIN, 1 }
    A:=345;                                    { MAIN, 2 }
    CH:='A';                                  { MAIN, 3 }
    ADD(1);                                    { MAIN, 4 }
    Writeln('I= ', I:3);                      { MAIN, 5 }
    FOR J:=1 TO 10 DO                          { MAIN, 6 }
        Writeln(' J=', J:2);                  { MAIN, 7 }
        Writeln(' .....END.....')           { MAIN, 8 }
    END.

```

FIG DEB04 PROGRAM

编成时生成的列表文件 A.LST 如 FIG DEB05 PROGRAM 所示。

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
1				PROGRAM A(INPUT, OUTPUT);
2				{ A.PAS.....DEBUGGER EXAMPLE 4 }
3				TYPE
4				ARR=ARRAY [1..10] OF INTEGER;
5				VAR
6				I, J:INTEGER; A:REAL; CH:CHAR; AR:ARR;
7				
8				PROCEDURE ADD (VAR I2:INTEGER);
9				VAR
10				J2, K2:INTEGER; AR2:ARR;
11				
12				FUNCTION SUB (I3:INTEGER):INTEGER;
13				VAR

14				J3:INTEGER;	
15				BEGIN	(• START OF SUB •)
16	1	3	1	J3:=100;	{ SUB, 1 }
17	2	3	1	SUB:=J3-I3;	{ SUB, 2 }
18	3	3	1	WRITELN('SUB=', (J3-I3):3)	{ SUB, 3 }
19				END;	(• END OF SUB •)
20					
21				BEGIN	(• START OF ADD •)
22	1	2	1	K2:=2;	{ ADD, 1 }
23	2	2	1	K2:=222;	{ ADD, 2 }
24	3	2	1	FOR J2:=1 TO 4 DO	{ ADD, 3 }
25	4	2	2	AR2 (J2) :=2 * J2;	{ ADD, 4 }
26	5	2	1	I2:=AR2 (1) +SUB(10);	{ ADD, 5 }
27	6	2	1	WRITELN ('I2=', I2:3)	{ ADD, 6 }
28				END;	
29					(• END OF ADD •)
30					
31				BEGIN	(• START OF MAIN •)
32	1	1	1	A:=3.1415;	{ MAIN, 1 }
33	2	1	1	A:=3.345;	{ MAIN, 2 }
34	3	1	1	CH:='A';	{ MAIN, 3 }
35	4	1	1	ADD (1) ;	{ MAIN, 4 }
36	5	1	1	WRITELN ('I= ', I:3);	{ MAIN, 5 }
37	6	1	1	FOR J:=1 TO 10 DO	{ MAIN, 6 }
38	7	1	2	WRITELN (' J=', J:2);	{ MAIN, 7 }
39	8	1	1	WRITELN ('END.....')	{ MAIN, 8 }
40				END.	

ERRORS DETECTED: 0

FREE MEMORY: 13977 WORDS

FIG DEB05 PROGRAM

在这一节中，我们反复介绍了这个程序，因为在下面详细介绍各调试命令时，要多次用到它。

这个程序由主程序 MAIN、过程 ADD 和层号为 3 的函数 SUB 组成。

主程序有八个语句，第一个语句 A:=3.1415，即对实数变量 A 赋以实数值 3.1415，以后这一语句用“MAIN, 1”表示。第二个语句是第二次对变量 A 赋值。第三个语句是对字符变量 CH 赋以字符 A。第四个语句 ADD (1)，表示调用过程 ADD，其中 1 表示实际参数，对应过程 ADD 中的变量形式参数 I2。开始的时候，实际参数 1 没有赋以初值，故为 0；而当调用过程结束时，变量形式参数的值为 92，并将此值传递给实际参数 I。所以在第五个语句输出 I 值时，它为 92。第六第七语句是输出全程变量 J 的值的循环语句。第八语句是当主程序结束时，输出字符串‘.....END.....’。

过程 ADD 共有六个语句，函数 SUB 共有三个语句。对于它们的每一条语句不作介绍了，读者很容易看懂。只是提出一个思考题给读者，函数中的第三个语句“SUB, 3”与过

程中的第六个语句“ADD, 6”，有什么不同？函数中的第三语句，明明是要输出函数SUB的值，但不能写 SUB:3，而要写成 (J3—I3):3，这是为什么？

这一章的内容，由于上机实习的要求，在未学习第五章之前，许多读者要提前学习。如果初学者对过程和函数不熟悉，在学习这部分调试程序时，对上述程序中的过程及其函数，可以暂时不要求理解，而只看主程序部分。而主程序的第四语句 ADD (1) 可暂时看作是 J:=92。但是如果要真正掌握调试程序，深刻理解各种调试命令的功能和使用，还应认真学习和掌握第五章以及其它各章的内容。

在详细介绍各种调试命令之前，我们先看一下就源程序 A.PAS，使用 DEBUGGER 调试程序，程序运行的结果

```
.RUN A
PASCAL DEBUGGER V2.2
LISTING FILE NAME? A
); P
SUB= 90
I2= 92
I= 92
J = 1
J = 2
J = 3
J = 4
J = 5
J = 6
J = 7
J = 8
J = 9
J = 10
.....END.....
Program termination at MAIN, 8
); C
```

(1) 断点命令

为了控制程序流程，可以通过设置断点，让程序停在指定的语句上。设置断点命令的一般格式如图 10.3-1 所示：

其中过程名包括主程序MAIN，各种层号的过程或函数的名称；语句号是指相应过程中的语句号。

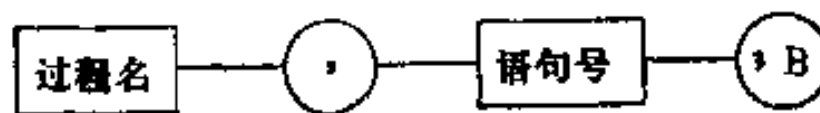


图 10.3-1 断点命令的一般格式

如果连续设置断点，只有最后设置的那个断点命令起作用。如果设置的断点在程序中根本不存在的话，并不影响程序的运行。

清除断点用 0: B 或 ; B 来实现。

退出调试（或称调试程序的出口）用 CTRL/C，只需要打一次 CTRL/C 键就可以从调试命令状态回到监控命令状态。

就上述 A.PAS 源程序使用调试程序，进入调试入口，然后进行如下操作，即可实现断点命令的设置与清除。程序运行如下：

<pre>] MAIN, 1; B] ; P Breakpoint at MAIN, 1] MAIN, 7; B] ; P SUB= 90 I2= 92 I= 92 Breakpoint at MAIN, 7] ; P J = 1 Breakpoint at MAIN, 7] ; P J = 2 Breakpoint at MAIN, 7] 0; B] ; P J = 3 J = 4 J = 5 J = 6 J = 7 J = 8 J = 9 J = 10END..... Program termination at MAIN, 8] ^C </pre>	<p>在主程序的第一语句设置断点；</p> <p>启动程序或继续运行；</p> <p>断点在主程序第一语句上；</p> <p>在主程序的第七语句设置断点；</p> <p>继续运行</p> <p>断点在主程序的第七语句上；</p> <p>继续；</p> <p>断点在主程序的第七语句上；</p> <p>继续</p> <p>断点在主程序的第七语句上；</p> <p>清除断点；</p> <p>程序结束在主程序的第八语句上</p> <p>退出调试，打 CTRL/C 键；</p> <p>回到监控命令状态。</p>
---	---

重新运行 A.SAV，进入调试程序入口，设置三个断点，操作如下：

<pre>] SUB, 1; B] ADD, 3; B] MAIN, 1; B] P; Breakpoint at MAIN, 1] </pre>	<p>在函数 SUB 的第一个语句设置断点；</p> <p>在过程 ADD 的第三个语句设置断点；</p> <p>在主程序的第一个语句设置断点；</p> <p>继续运行</p> <p>断点在主程序的第一个语句上</p>
--	---

运行结果表明，最后设置的断点起作用。

(2) 单步命令

设置单步和清除单步的命令如下:

N ; S

设置单步的命令, N为非零整数;

0 ; S 或; S

清除单步的命令。

设置单步方式以后, 调试程序在每一个语句执行前停止程序, 如同那里设置了断点。用 ; P 命令一次执行程序一个语句。可以用一个正整数 n 和 ; P 命令一起通知程序执行 n 个语句。

对初学者来说, 单步方式有着特殊的意义, 它在其它调试命令配合下, 可以一步一步地观察一个程序是如何运行的, 帮助用户透彻地了解程序的运行过程。

下面举例说明单步命令的应用。重新运行 A.SAV, 进入调试程序入口,

] 3 ; S

设置单步方式;

] ; P

继续运行;

Breakpoint at MAIN, 1

断点在主程序第一个语句上;

] ; P

继续;

Breakpoint at MAIN, 2

断点在主程序第二个语句上;

] ; P

继续;

Breakpoint at MAIN, 3

断点在主程序第三个语句上;

] 0 ; S

清除单步;

] ; P

继续运行;

SUB= 90

I2= 92

I = 92

J = 1

J = 2

J = 3

J = 4

J = 5

J = 6

J = 7

J = 8

J = 9

J = 10

.....END.....

Program termination at MAIN, 8

程序在主程序第八个语句上结束;

] ^C

退出调试;

回到监控命令状态。

(3) 启动或继续命令

; G 和; P 这两个命令都是用于开始执行程序, 或从断点继续执行程序的命令。

如果已经设置了断点, 则 n ; P 或 n ; G 命令可以使程序通过断点 n 次。

如果已经设置了单步方式, 则 n ; P 或 n ; G 命令使程序执行 n 个语句。

具体运行在上面已经有反映, 为更清楚起见, 再说明如下。重新运行 A.SAV, 进入调

试程序入口,

```
    ) MAIN, 1; B
    ); G
    Breakpoint at MAIN, 1
    ) 2; S
    ); P
    Breakpoint at MAIN, 2
    ) 2; P
    Breakpoint at MAIN, 4
    ) 0; S
    ) MAIN, 7; B
    ); P
    SUB= 90
    l2= 92
    l= 92
    Breakpoint at MAIN, 7
    ); P
    J = 1
    Breakpoint at MAIN, 7
    ); P
    J = 2
    Breakpoint at MAIN, 7
    ) 7; G
    J = 3
    J = 4
    J = 5
    J = 6
    J = 7
    J = 8
    J = 9
    Breakpoint at MAIN, 7
    )
```

以上三个调试命令是属于控制程序流程的命令。

(4) 列出可用的变量名的命令

用 \$V 命令, 可以列出当前有效的变量的名称。打印出的表从当前活动的过程名开始, 后面是它的局部变量, 打印时从行首缩进三个空格。然后打印调用当前活动的过程的过程名, 如此等等, 沿着堆栈一直打印出全程变量。如果变量很多, 按下 CTRL/C 键即停止继续打印或显示变量, 再打 CTRL/C 键, 系统回到监控命令状态。在列变量过程中, 按一次 CTRL/C 键, 则变量名称列完回到监控命令状态。若连续按二次 CTRL/C 键, 则立即回到监控命令状态。

在主程序的第一个语句设置断点;

继续运行;

断点在主程序的第一个语句上;

设置单步方式;

继续运行;

断点在主程序的第二个语句上;

使程序执行二个语句;

断点在主程序的第四个语句上;

清除单步方式;

在主程序的第七个语句设置断点;

继续;

断点在主程序的第七个语句上;

继续;

断点在主程序的第七个语句上;

继续;

断点在主程序的第七个语句上;

使程序通过断点七次;

断点在主程序的第七个语句上。

重新运行 A.SAV, 进入调试程序入口;

) MAIN, 1; B

) ; P

Breakpoint at MAIN, 1

) \$ V

MAIN

I

J

A

CH

AR

) ADD, 1; B

) ; P

Breakpoint at ADD 1

) \$ V

ADD

I2

J2

K2

AR2

MAIN

I

J

A

CH

AR

) SUB, 1; B

) ; P

Breakpoint at SUB

) \$ V

SUB

I3

J3

ADD

I2

J2

K2

AR2

MAIN

I

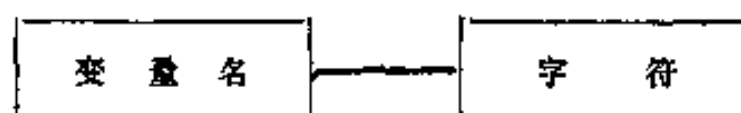
```

]
A
CH
AR
]

```

(5) 查看变量内容的命令

查看变量内容的命令格式如下：



其中字符与变量的类型有关，其对应关系是：

显示方式 (变量类型)	字 符
十 进 制 整 数	/
十 进 制 字 节	\
实 数	%
字 符	'

其中字节方式也用于查看枚举类型或布尔类型的变量，字符数据也可以作为字节数据查看。布尔变量取真 (TRUE)，字节为 1；布尔变量取假 (FALSE)，字节为 0。用字节查看字符类型变量，得到的是字符的字符码。

查看变量内容的命令，只能查看那些目前有效的变量，即只能查看层号 (LEVEL) 数相同或比它低的层号的那些过程中的变量。也就是说，断点设在主程序中，只能查看全程变量，不能查看它调用的过程中的局部变量；断点设在层号为 2 的过程中，则可以查看层号为 2 的那个过程中的局部变量，不能查看层号为 3 的过程中的局部变量，但可以查看主程序中的全程变量的内容。

重新运行 A.SAV，进入调试程序的入口：

```
] SUB, 3; B
```

在层号为 3 的函数 SUB 第三语句设置断点；

```
] ; P
```

继续运行；

```
Breakpoint at SUB, 3
```

断点在函数 SUB 的第三语句上；

```
] J3/
```

查看整型变量 J3 的内容；

```
100
```

```
] K2/
```

查看层号为 2 的过程 ADD 中的整型变量 K2；

```
222
```

```
] A %
```

查看层号为 1 的主程序中的实型变量 A；

```
7.345000E +00
```

```
] ADD, 6; B
```

在层号为 2 的过程 ADD 的第 6 语句设置断点

```

} : P
SUB= 90
Breakpoint at ADD, 6
} J3/
Can't find variable called "J3"
16875
} AR2/
2
} AR2/4
.+0/2
.+2/4
.+4/6
.+6/8
} AR2+4/
6
} AR2.4/
6
} AR2.8-4/
6
} MAIN, 6; B
} : P
I2= 92
I = 92
Breakpoint at MAIN, 6
} J3/
Can't find variable called "J3"
-12896
} K2/
Can't find variable called "K2"
-12896
} A %
7.345000E+00
} CH'
A
} CH\
65
}

```

(6) 修改变量值的命令

在调试程序的过程中，用 PASCAL 的赋值运算符:= 就可以给程序中的变量赋以新的内容，以达到改变变量值的目的。

继续运行：

断点在过程 ADD 的第六语句上；
查看层号比 ADD 高的函数 SUB 中变量 J3；
显示找不到这个变量 J3；

查看主程序中的变量数组 AR2；

查看全程变量数组 AR2 的四个值；

AR2 [1] 的值为 2；

AR2 [2] 的值为 4；

AR2 [3] 的值为 6；

AR2 [4] 的值为 8；

查看 AR2 [3] 方法 1；

查看 AR2 [3] 方法 2；

查看 AR2 [3] 方法 3；

在层号为 1 的主程序第六语句设置断点；
继续；

断点在主程序第六语句上；

查看层号为 3 的函数 SUB 中的变量 J3；
显示找不到这个变量；

查看层号为 2 的过程 ADD 中的变量 K2；
显示找不到这个变量；

查看主程序中的变量 A；

查看主程序中的变量 CH；

查看主程序中的变量 CH 的字符码；
显示字符 A 在 ASCII 码中对应的字符码。

例如:

A:=1; 给整型变量A赋以整数1;
B:=R1.34; 给实型变量B赋以实数1.34;
C:='A'; 给字符型变量C赋以字符'A'。

请注意, 只能对目前有效的变量赋值。

重新运行 A.SAV, 进入调试程序入口;

] ADD, 3; B

在过程 ADD 的第三语句设置断点;

] : P

继续运行;

Breakpoint at ADD, 3

断点在过程 ADD 的第三语句上;

] K2/

查看过程ADD中的局部变量 K2;

222

] K2:=333

给变量K2赋以整数333;

] K2/

再查看变量K2的内容;

333

] A %

查看主程序中实型变量A;

7.345000 E+00

] A:=R1.234

给变量A赋以实数1.234;

] A %

再查看变量A;

1.234000 E+00

] A:=R5.678000 E+00

再给变量A赋以实数5.678000E+00;

] A %

再查看变量A的内容;

5.678000 E+00

] CH'

查看主程序中字符变量CH;

A

] CH:='B'

给字符变量CH赋以字符'B'

] CH'

再查看变量CH。

B

]

(7) 监视变量命令

监视命令用于监视一个变量的值, 该变量的值改变时, 程序就中断并停在修改变量语句的后面。

对一个变量设置监视命令, 只要打入变量名, 后面跟以;W。而清除监视, 则打入命令, -1;W。

监视变量命令, 只对目前有效的变量进行监视。监视局部变量, 要十分小心, 只有当定义它们的过程或函数执行时才能使用, 否则会出现错误信息。为此, 首先在定义该变量的过程或函数的第一个语句上设置断点, 等待程序执行到那里时, 方可对这个局部变量设置监视。

重新运行 A.SAV, 进入调试程序入口;

] MAIN, 2; B

在主程序第二个语句设置断点;

] : P

继续运行;

Breakpoint at MAIN, 2

] A %

3.141500E+00

] A; W

] ; P

Watched value changed.

Breakpoint at MAIN, 3

] A %

7.345000E+00

] K2; W

Can't find variable called "K2"

] ADD, 2; B

] ; P

Breakpoint at ADD, 2

] K2/

2

] K2; W

] ; P

Watched value changed.

Breakpoint at ADD, 3

] K2/

222

]

断点在主程序第二个语句上;

查看主程序的全程变量A;

对变量A设置监视命令;

显示程序监视着变量值的改变;

断点在主程序第三个语句上;

再查看变量A;

对过程ADD中的局部变量设置监视;

不能监视过程ADD中的变量K2;

在过程ADD的第二语句设置断点;

断点在过程ADD的第二语句上;

查看过程中局部变量K2;

对局部变量K2设置监视命令;

显示程序监视着变量K2值的改变;

断点在修改变量值语句的后面;

查看变量K2的值;

(8) 查看变量地址的命令

如果需要知道某个变量(或数据记录)的地址,可使用操作符 "=", 打印变量的地址而不是它的内容。命令格式如图 10.3-2 所示:



图 10.3-2 查看变量地址命令的格式

执行此命令后,显示变量的绝对地址,用十进制数表示。

使用绝对地址要小心。对于全程变量来说,它们自始至终都存在,因为它们是静态地分配内存单元的,所以有绝对地址。对于局部变量来说,只有在该过程或函数投入运行时才被使用,在过程投入运行前并不分配内存单元,在投入运行时才分配内存单元。所以局部变量是动态地分配内存单元,也就是说局部变量的地址不是绝对静止不变的,而是随着它所在的过程的不同执行而改变的。这样,读者也就不难理解下面的问题:断点设在层号低的过程时,不能去查看层号比它高的过程中的局部变量。另外,也不要去查看调试程序已被报告找不到的变量,这些变量已从堆栈上撤消,其它变量可能已占用它们原来的位置。

对于下面应用举例中遇到的负地址问题先说明一下:对于 PDP-11/03 系统来说,内存容量为32K字(字长16位),精确地说应是32768个字内存单元。用绝对地址表示将其地址

分为=截, 从 00000 单元到 16383 单元 (即八进制的 000000 B 到 077777 B) 用正整数表示, 另一半 (即八进制的 100000 B 到 177777 B) 用负整数 (-00000 到 -16383) 表示。

重新运行 A.SAV, 进入调试程序的入口

] MAIN, 1; B

] ; P

Breakpoint at MAIN, 1

] I =

13324

] J =

13326

] A =

13328

] CH =

13332

] AR =

13334

] I 2 =

Can't find variable called "I2"

0

] J 3 =

Can't find variable called "J3"

0

] ADD, 1; B

] ; P

Breakpoint at ADD, 1

] I 2 =

-12888

] J 2 =

-12914

] K 2 =

-12912

] I 3 =

Can't find variable called "J3"

0

] J 3 =

Can't find variable called "J3"

0

] SUB, 1; B

] ; P

Breakpoint at SUB, 1


```

) I 3 =
-12918
) J 3 =
-12922
) J 2 =
-12888
) J 2 =
-12914
) K 2 =
-12912
) I =
13324
) J =
13326
) A =
13328
) CH =
13332
) AR =
13334
)

```

(9) 列出程序最后执行的十个语句的命令

PASCAL 调试程序总是记住最后执行的十个语句。如果需要知道程序运行如何到达断点, 或者想知道什么语句改变了被监视变量的值, 可以使用 \$L 命令对最后执行的十个语句的位置列表。

重新运行 A.SAV, 进入调试程序的入口,

```

) ADD, 1; B
) ; P
Breakpoint at ADD, 1
) $L
MAIN, 1
MAIN, 2
MAIN, 3
MAIN, 4
ADD, 1
) SUB, 3; B
) ; P,
Breakpoint at SUB, 3
) $L
ADD, 2

```

```

ADD, 3
ADD, 4
ADD, 4
ADD, 4
ADD, 4
ADD, 5
SUB, 1
SUB, 2
SUB, 3
]

```

(10) 跟踪命令

跟踪命令的格式是:

N T 置跟踪方式, N为非零数

0 T或 ;T 清除跟踪方式.

置跟踪方式, 执行跟踪命令, 调试程序打印出或在终端上显示出启动后要执行的全部语句. 这样用户可以监视程序的执行, 以便随着控制流程发现各种问题.

重新运行 A.SAV, 进入调试程序入口:

```

] MAIN 1; B
] ; P
Breakpoint at MAIN, 1
] 1; T
] MAIN 4 B
] ; P
MAIN, 2
MAIN, 3
Breakpoint at MAIN, 4
] MAIN, 5; B
] ; P
ADD 1
ADD 2
ADD 3
ADD 4
ADD, 4
ADD 4
ADD 4
ADD 5
SUB, 1
SUB, 2
SUB, 3
SUB= 90

```

```

ADD, 6
I2= 92
Breakpoint at MAIN, 5
) ; T
) ; P
I =92
  J = 1
  J = 2
  J = 3
  J = 4
  J = 5
  J = 6
  J = 7
  J = 8
  J = 9
  J =10
  .....END.....,
Program termination at MAIN, 8
) ^C
.

```

(11) 跟踪过程调用命令

跟踪过程调用命令的格式:

N; C 置跟踪过程调用方式, N为非零数

0 ; C 或 ; C 清除跟踪过程调用方式。

执行跟踪过程调用命令时, 调试程序会打印出或在终端上显示启动后过程或函数的调用情况。

在许多情况下, 跟踪整个程序需要很多时间和纸张 (如果没有显示终端), 为了解决这一问题, 采用跟踪过程调用的方式, 每次进入或退出过程, 调试程序就会通知用户。这样, 可以在逐个执行过程的基础上监视程序的进行。

重新运行 A.SAV, 进入调试程序入口,

```

) 1; C
) ; P
Entering ADD from MAIN, 4
Entering SUB from ADD, 5
  SUB= 90
Leaving SUB
I2= 92
Leaving ADD
I= 92
  J = 1

```

```

J = 2
J = 3
J = 4
J = 5
J = 6
J = 7
J = 8
J = 9
J = 10
.....END.....
Program termination at MAIN, 8
) ^C

```

(12) 过程执行堆栈列表命令

如果程序被断点或监视陷阱所中断，用户想知道程序如何到达中断的语句，可以用 \$ S 命令。观察过程执行堆栈中的内容。这个命令首先列出当前过程或函数的名称，执行调用或操作的语句号。然后列出调用该过程或函数的层号较低的过程或函数的名称及其调用语句，一直到调用过程或函数的主程序及其相应的语句。

重新运行 A.SAV，进入调试程序入口：

```

) ADD, 1; B
) : P
Breakpoint at ADD, 1
) $ S
ADD, 1
MAIN, 4
) SUB, 1; B
) : P
Breakpoint at SUB, 1
) $ S
SUB, 1
ADD, 5
MAIN, 4
) : P
SUB= 90
I2= 92
I= 92
J = 1
J = 2
J = 3
J = 4

```

J = 5

J = 6

J = 7

J = 8

J = 9

J = 10

.....END.....

Program termination at MAIN, 8

J ^ C

第十一章 综合应用程序举例

综合应用程序举例是本书最后一章的内容，共有十二个程序，都有一定的实用意义，对读者有重要的参考价值。

这一章的目的，为的是让读者比较全面地复习和应用 PASCAL 语言的基本概念及其程序设计技术，进一步提高综合应用的能力。对解决有关专业中的类似问题，也有较大的参考价值。

第一、第二个程序，是解决作图问题。按求解结果或某些标准函数画曲线，是许多读者关心的课题，也是科技工作者经常遇到的问题。

第三个程序，是介绍对随机给出的一串整数用几种方法进行排序，并对各种排序方法进行比较。排序是从事软件的工程人员的基本功。

第四个程序，它是解决实验心理学中一个古典的迷宫路径问题。

第五个程序是化工专业的液体混合物导热系数及其误差的计算问题，在这个程序中使用了最优化计算方法。

第六、第七个程序，介绍了解决同一类型问题的两种不同方法，这对于从事经济、行政管理的人来说可以从中得到启发。读者要解决统计工作的管理程序，往往要遇到处理几个不同类型的数据文件。第六个程序是解决此类问题的一个典型程序，作为一个实例，这里仅处理两个较短的数据文件。但是解决问题的基本原理和程序设计方法有普遍性，读者可以举一反三，去解决包含几个、几十个不同类型的数据文件的复杂问题。在实际工作中，往往是在一个数据文件中包括整数、实数、字符等不同类型的数据。第七个程序介绍的正是如何处理这样的疑难问题。

第八个程序，是解决系统工程中的关键路径问题。

第九、第十个程序，是介绍另一类管理问题。第九个程序介绍了气泡沉浮法。第十个程序在数组设置和读取文件技巧上有独到之处。

第十一程序，实现的是布尔函数的化简问题，这对从事计算机设计的读者，提供了用计算机来化简布尔函数的一种方法。

第十二个程序，介绍了人和计算机进行下棋比赛的游戏。在这个程序中，除了记录和指针类型数据未用到外，其它各种数据类型都用到了，这是一个综合性较强的程序。

之所以没有列举一般的科学计算的程序。这是因为这类问题已在前面各章中作过介绍，读者也比较容易掌握。

在这十二个程序中，每一个都用到了文件，反复利用了过程，可见文件和过程的概念及其应用是何等的重要。

在介绍具体的程序之前，需要说明以下两点：

第一，本章给出的程序，涉及到某些专业性较强的知识，比如数据结构、布尔函数化简、液体混合物导热系数及误差统计等等。对这些专业知识，本书不作深入的介绍，只是概括地提示一下，读者对与自己无关的专业内容，可以不作深入的研究，但可以从解决课题的方法上，程序设计的技巧上吸取有用之处。

第二,限于篇幅,本章不准备对给出的每一个程序从头至尾详细地加以说明,只是对每一个程序的功能和基本结构作简单介绍。大部分程序,经过仔细阅读是能够看得懂的,个别程序的某些部分,少数读者感到有一定困难,这是很正常的。因为每一个程序设计者,有自己的设计思想和风格,要阅读其他人编写的程序,强制自己去领会自己不熟悉的设计思路,这本来是一件不容易做到的事。既然这些程序仅供参考,对一些难度较大的程序,可只作为基本要求。读者可以根据自己的需要,选择地加以利用,并在熟习的基础上予以改进。因为这些应用程序还有很多可以改进和提高的地方。

通过这些程序的阅读、模仿、改进或自行编制,有助于读者复习有关的基本概念,熟悉程序设计技巧,这对掌握 PASCAL 程序设计语言和提高综合应用水平,不仅是十分必要的,而且是十分有益的。

第一个程序 (TOTAL1.PAS)

功能:在同一坐标上画出三种标准函数 ($Y=\sin(X)$, $Y=\sin(2X)$ 和 $Y=\sin(X)+\sin(2X)$) 的曲线图,并且用特殊的符号表示曲线的交点。

本程序用字符元素类型数组记忆对应的各标准函数的曲线坐标点,这是该程序的特点。

如果用其它的方法,而不设置字符类型数组是否也可以实现呢?如果只是画一条函数曲线,当然是可以的。但若同时要画几个函数的曲线,不用字符元素类型数组而用普通的方法是很难实现的。

另外,本程序在设计运算结果的表头,X轴和Y轴的问题上是经过仔细考虑的。不要说粗枝大叶,即使认真考虑,稍不注意也会出差错。请注意,该程序的输出数据文件的坐标原点,它并不是根据计算结果而作图的。程序开始作图的起点是X的值为 9° (即 $9 \cdot \pi/180$),而不是从 0° 开始的。为了便于阅看函数曲线,在曲线前后都附有必要的文字说明。这些做法的好处请读者在实践中去体会。源程序如下。

```
PROGRAM TOTAL1(OUTPUT);
(* GRAPH OF SIN(X), SIN(2X) AND SIN(X)+SIN(2X) *)
CONST
    PI=3.14159;
VAR
    I, J, N1, N2, N3, N: INTEGER;
    X, Y1, Y2, Y3, Y: REAL;
    F: ARRAY [0..80] OF CHAR; M: TEXT;
    PROCEDURE INDEX(Y: REAL);
    BEGIN
        N:=TRUNC(Y*15)+30;
    END;
BEGIN (* MAIN PROGRAM *)
    REWRITE(M, 'TOTAL1.DAT');
    WRITELN(M, 'Y= SIN(X), Y= SIN(2X), Y= SIN(X)+SIN(2X)');
    WRITELN(M);
    WRITE(M, 'X: 0 10 20 30 40 50 60 70 80');
    WRITELN(M, 'Y: -2 -1 0 1 2');
```

```

WRITE(M, '-----+-----=');
WRITELN(M, '-----+-----+-----');
FOR I:=1 TO 60 DO F[I] := ' ',
X:=9*PI/180;
FOR I:=1 TO 40 DO
  BEGIN
    F[39] := 'I'; Y1:=SIN(X);
    Y2:=SIN(2*X); Y3:=SIN(X)+SIN(2*X);
    INDEX(Y1); N1:=N; F[N1] := '+';
    INDEX(Y2); N2:=N;
    IF N2=N1
      THEN F[N2] := '=' ELSE F[N2] := 'O';
    INDEX(Y3); N3:=N;
    IF (N3=N2) OR (N3=N1)
      THEN F[N3] := '=' ELSE F[N3] := '*';
    FOR J:=0 TO 60 DO WRITE(M, F[J]);
    WRITELN(M);
    F[N1] := ' '; F[N2] := ' '; F[N3] := ' '; X:=X+9*PI/180
  END;
WRITELN(M); WRITELN(M);
WRITELN(M, ' :18, " + "-----Y=SIN(X)');
WRITELN(M, ' :18, " O"-----Y=SIN(2X)');
WRITELN(M, ' :18, " * "-----Y=SIN(X)+SIN(2X)');
WRITELN(M, ' :18, " = "-----CROSSPOINT OF CURVE');
CLOSE(M)

```

END

输出数据文件请看下页 FIG TOTAL1 DATA.

第二个程序 (TOTAL2.PAS)

功能：把 ADP 电光调制器的最佳补偿角 Y 与二晶片双折射系数之差 X 的函数关系，用图表示。

$$\text{已知: } Y = \frac{K + \sqrt{K^2 + A}}{B},$$

$$\text{其中 } K = \frac{M \cdot L}{N_1} \cdot X;$$

$$A = L \cdot P \cdot X, \quad B = L \cdot X;$$

$$M = N^2 \cdot \left(\frac{1}{NO^2} - \frac{1}{NE^2} \right) \cdot \sin(Z) \cdot \cos(Z);$$

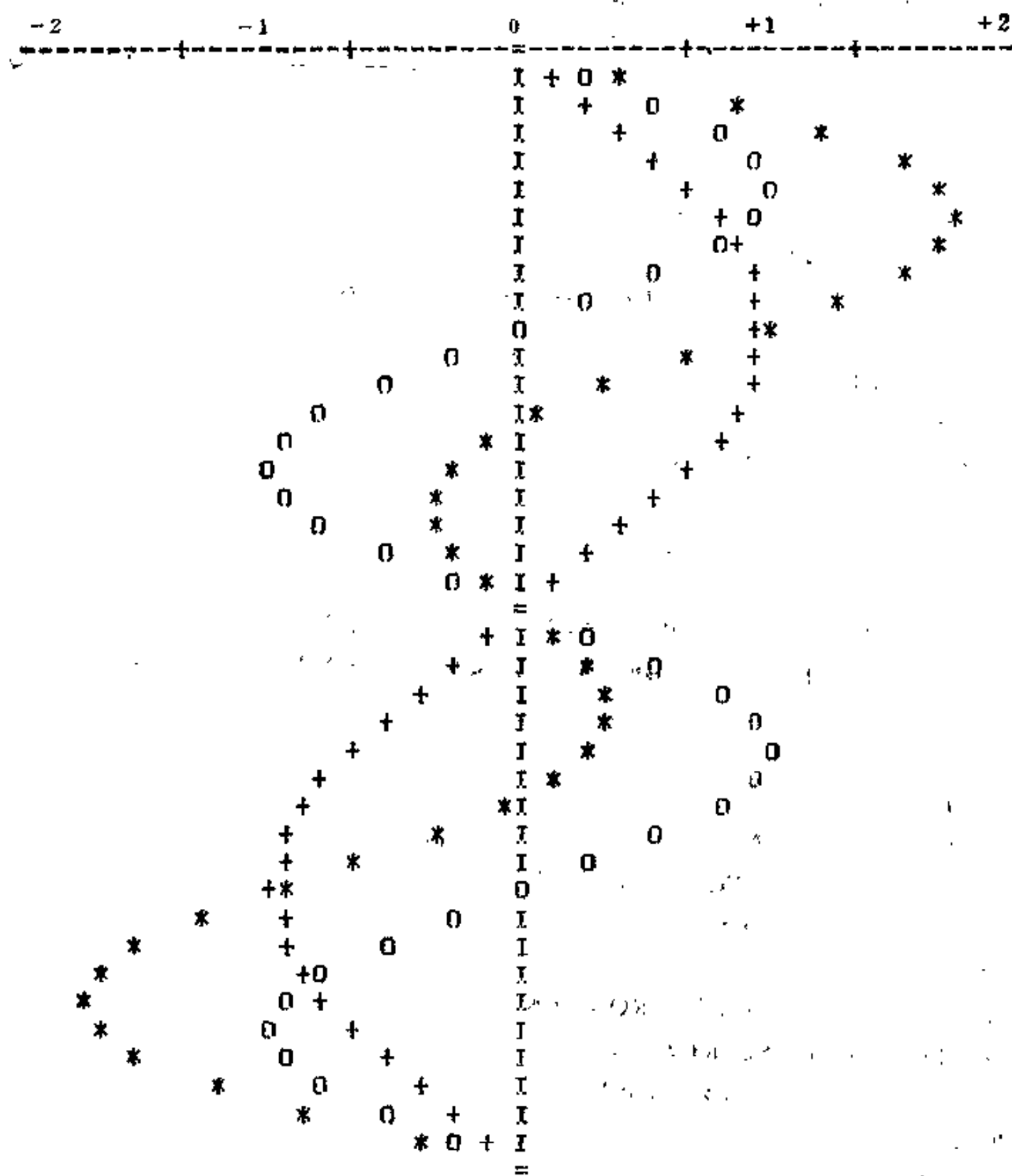
$$\frac{1}{N^2} = \frac{\cos^2(Z)}{NO^2} + \frac{\sin^2(Z)}{NE^2};$$

$$N_1 = NO - N;$$

$$Z_1 = 37^\circ, \quad Z_2 = 45^\circ, \quad Z_3 = 90^\circ;$$

OUTPUT, (• TOTAL1.DAT •)

$Y = \sin(X)$ $Y = \sin(2X)$ $Y = \sin(X) + \sin(2X)$



'+' ----- $Y = \sin(X)$
 '0' ----- $Y = \sin(2X)$
 '*' ----- $Y = \sin(X) + \sin(2X)$
 '=' ----- CROSSPOINT OF CURVE

FIG TOTAL1 DATA

本程序中，函数运算用过程 OCA 完成，作图在主程序中进行，用字符元素类型数组 F 来记忆对应的函数关系的坐标点。

该程序的 X 坐标变化是按一定倍数 (1.2 倍) 递增的。图中注上的 X 和 Y 数值是根据在辅助调试程序帮助下单步运行时查得的。

PASCAL 源程序和输出曲线如 FIG TOTAL2 PROGRAM 所示。

```

PROGRAM TOTAL2(INPUT,OUTPUT);
{ GRAPH OF OPTIMUM COMPONSATION ANGLE AS A FUNCTION
  OF DIFFERENCE OF BIREFRACTIVE INDICES }
CONST
  PI=3.14159;
VAR
  I,J,T,T1,T2,T3      : INTEGER;
  X,L,P,Y,Z1,Z2,Z3    : REAL;
  F                    : ARRAY [0..65] OF CHAR;
  M                    : TEXT;
  PROCEDURE OCA(Z:REAL);
  CONST
    NO=1.521,    NE=1.476;
  VAR
    M,N1,N A B K      : REAL;
  BEGIN
    N:=1/SQRT(SQR(COS(Z)/NO)+SQR(SIN(Z)/NE));
    M:=N * SQR(N) * (1/SQR(NO)-1/SQR(NE)) * SIN(Z) * COS(Z);
    N1:=NO-N;
    A:=L * P * X;
    B:=L * X;
    K:=M * L * X/N1;
    Y:=(K+SQRT(SQR(K)+A))/B;
    T:=TRUNC(Y * 50)
  END
BEGIN      (* MAIN PROGRAM *)
  Writeln('PLEASE READ L,P');
  Rewrite(M,'TOTAL2','DAT');
  Readln(L,P);
  X:=1.0E-5;
  FOR I:=1 TO 55 DO WRITE(M '- ');
  Writeln(M);
  FOR I:=0 TO 65 DO F[I] := ' ';
  FOR I:=0 TO 30 DO
  BEGIN
    F[0] := '!';
    Z1:=PI * 37/180;
    Z2:=PI/4;

```

```

Z3:=PI/2;
OCA(Z1) T1:=T; F (T1) :=' * '
OCA(Z2) T2:=T; F (T2) :=' '
OCA(Z3) T3:=T; F (T3) :=' + '
FOR J:=0 TO 65 DO WRITE(M,F(J));
WRITELN(M);
F (T1) :=' ' , F (T2) :=' ' , F (T3) :=' ' ;
X:=1.2 * X
END,
CLOSE(M)
END
NOTE : L.....LENGTH OF CRYSTAL SLAB ( 6 cm)
      P.....LIGHT WAVELENGTH 6.328E-05 cm)

```

OUTPUT:

TOTAL2.DAT

*)

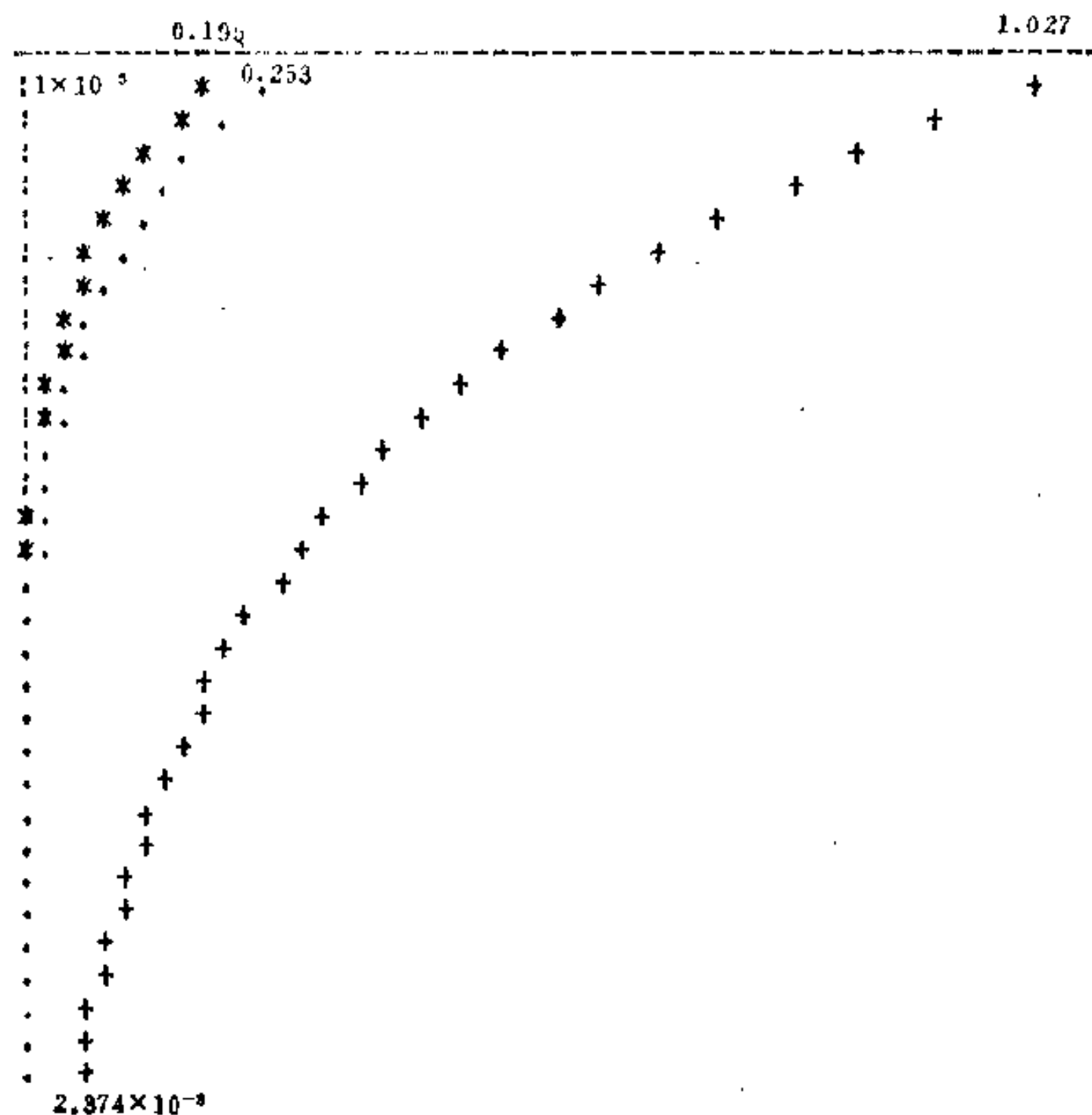


FIG TOTAL2 PROGRAM

第三个程序 (TOTAL3.PAS)

功能：对随机给出的一串整数用五种方法进行排序，并显示排序结果和记录排序时间。

排序是计算机科学中很重要的一个子领域。在各类软件工程中，各种排序方法是必不可少的，也是从事软件的工程人员所必须掌握的。根据记录所处的位置，排序又分为内部排序和外部排序两种。在排序期间全部数据都存放在内存的算法，称为内部排序。如果数据量太大，全部记录不可能同时存放在内存，排序期间记录要在内外存之间移动，这种排序称之为外部排序。

本程序介绍的是内部排序，用五种方法对随机给出的一串整数进行排序。这五种方法是

- (1) 插入法；
- (2) 折半法；
- (3) 堆排序法；
- (4) 快速排序法；
- (5) 归并排序法。

在程序中这五种排序方法是用五个过程来实现的，它们分别是 INSERTSORT, BINARYSORT, HEAPSORT, QUICKSORT 和 MERGESORT。

程序中产生随机整数用的是函数 RANDOM，输出排序结果用的是过程 OUT。

为节省起见，只输出1000个排序整数中的前十五个数。在输出文件中，只在第一个插入排序法的结果中，写出十五个整数，其余用省略表示，那是为避免重复和节约纸张而这样做的。实际输出并非如此。排序时间用秒表示。

排序的结果和排序时间送到文件 TOTAL3.DAT 中保存，以供随时显示或打印。

从输出结果来看，五种排序方法中以快速排序法效果最好，排序时间最省。

本程序的内容，以及程序运行结果详见 FIG TOTAL3 PROGRAM。

```
PROGRAM TOTAL3(INPUT,OUTPUT);
TYPE
  RNG=0..65535;
  AC=ARRAY [0..1000] OF RNG;
VAR
  A,B:AC;
  I,J,L,R,M,N:INTEGER;
  T1,T2:REAL;
  SEED,X:RNG; CR:TEXT;
  PROCEDURE INSERTSORT(VAR B:AC);
  BEGIN
    T:=TIME;
    FOR I:=2 TO N DO
      BEGIN
        X:=B[I]; B[0]:=X; J:=I-1;
        WHILE X<B[J] DO
          BEGIN
            B[J+1]:=B[J]; J:=J-1;
          END;
        B[J+1]:=X;
      END;
    T2:=TIME-T;
  END;
```

```

        B (J+1) :=X
    END,
    T2 :=TIME,
END,
PROCEDURE OUT(T1,T2:REAL, B:AC),
BEGIN
    FOR I:=1 TO 15 DO
        WRITELN(CR B (I) :5);
        WRITELN(CR);
        WRITE(CR,'TIME =',(T2-T1)*3600.0,'SECOND'),
        WRITELN(CR); WRITELN(CR);
    END,
PROCEDURE BINARYSORT(VAR B:AC),
BEGIN
    T1:=TIME,
    FOR I:=2 TO N DO
        BEGIN
            X:=B (I) ; L:=1;R:=1-1;
            WHILE L<=R DO
                BEGIN
                    M:=(L+R) DIV 2;
                    IF X<B (M)
                        THEN R:=M-1
                        ELSE L:=M+1
                END,
            FOR J:=1-1 DOWNT0 L DO    B (J+1) :=B (J) ,
            B (L) :=X
        END,
    T2:=TIME
END,
PROCEDURE HEAPSORT(VAR B:AC),
VAR
    L1,R1,R2:INTEGER
    X:RNG
PROCEDURE SIFT(L,R:INTEGER),
LABEL
    11,
VAR
    I,J:INTEGER
BEGIN    (• PRO.....SIFT •)
    I:=L; J:=2•I; X:=B (I) ;
    WHILE J<=R DO
        BEGIN
            IF J<R

```

```

    THEN
    IF B (J) < B (J+1)
    THEN J := J+1;
    IF X >= B (J)
    THEN GOTO 11;
    B (I) := B (J) , I := J, J := 22 * I
END;
11: B (I) := X
END;
BEGIN ( * PRO..... HEAPSORT * )
T1 := TIME;
FOR L1 := (N DIV 2) DOWNTO 1 DO
SIFT(L1,N)
FOR R1 := N DOWNTO 2 DO
BEGIN
X := B (1) ,
B (1) := B (R1) ,
B (R1) := X;
R2 := R1 - 1;
SIFT(1,R2)
END;
T2 := TIME
END;
PROCEDURE QUICKSORT(VAR B:AC);
PROCEDURE QUICK(L,R:INTEGER);
VAR
I,J1:INTEGER
X1:RNG;
BEGIN ( * PRO.....QUICK * )
IF P > L
THEN
BEGIN
I1 := L;
J1 := P;
X1 := B (I1) ,
REPEAT
WHILE (B (J1) > X1) AND (J1 > I1) DO
J1 := J1 - 1;
IF I1 < J1
THEN
BEGIN
B (I1) := B (J1) ,
I1 := I1 + 1;
END

```

```

        WHILE (B (I1) < X1) AND (J1 > I1) DO I1 := I1 + 1;
        IF I1 < J1
        THEN
            BEGIN
                B (J1) := B (I1)
                J1 := J1 - 1;
            END
        UNTIL I1 = J1;
        B (I1) := X1; I1 := I1 + 1; J1 := J1 - 1;
        QUICK(L, J1);
        QUICK(I1, P)
    END
END,
BEGIN      ( •   PRO.....QUICKSORT   • )
    T1 := TIME;
    QUICK(1, N);
    T2 := TIME
END
PROCEDURE MERGESORT(N:INTEGER, VAR A:AC);
VAR
    L, N1, M1, P:INTEGER
    PROCEDURE MEGPASS(L, N:INTEGER, VAR A, B:AC);
    VAR
        I:INTEGER
        PROCEDURE MERGE(P, M, N1:INTEGER, VAR A:AC);
        VAR
            J, K:INTEGER;
        BEGIN      ( •   PRO.....MERGE   • )
            I := P; K := P; J := M1 + 1;
            WHILE (I <= M1) AND (J <= N1) DO
                BEGIN
                    IF A (I) <= A (J)
                    THEN
                        BEGIN
                            B (K) := A (I); I := I + 1;
                        END
                    ELSE
                        BEGIN
                            B (K) := A (J); J := J + 1;
                        END;
                    K := K + 1;
                END;
            IF I > M1
            THEN FOR I := J TO N1 DO

```

```

        B (K+1-J) := A (I)
    ELSE FOR J:=1 TO M1 DO
        B (K+J-1) := A (J)
    END
BEGIN      ( •      PRO-----MERGEPASS      • )
    I:=1,
    WHILE I<=(N-2*L+1) DO
        BEGIN
            MERGE(I,I+L-1,I+2*L-1,A),
            I:=I+2*L
        END,
    IF (I+L-1)<N
    THEN
        MERGE(I,I+L-1,N,A)
    END,
BEGIN      ( •      PRO-----MERGESORT      • )
    T1:=TIME,
    L:=1
    SEED:=1,
    WHILE L<N DO
        BEGIN
            IF ODD(SEED)
            THEN MEGPASS(L,N,A,B)
            ELSE MEGPASS(L,N,B,A),
            L:=2*L,
            SEED:=SEED+1
        END
    T2:=TIME
    END,
FUNCTION RANDOM:RNG,
BEGIN
    SEED:=(SEED*13077+6925) MOD 32768,
    RANDOM:=SEED
END,
BEGIN      ( •      MAIN PROGRAM      • )
    WRITELN('PLEASE INPUT N= '),
    REWRITE(CR,'TOTAL3','DAT'),
    READLN(N),
    WRITELN(CR,'N=',N:5),
    WRITELN('OK'),
    FOR I:=1 TO N DO
        BEGIN
            A (I) :=RANDOM,
            B (I) :=A (I)

```



```

END.
WRITELN(CR,'### INSERT SORT ###');
WRITELN(CR);
INSERTSORT(B);
WRITELN('1');
OUT(T1,T2,B);
FOR I:=1 TO N DO B (I) :=A (I) ;
WRITELN(CR,'### BINARY SORT ###');
WRITELN(CR);
BINARYSORT(B);
WRITELN('2');
OUT(T1,T2,B);
FOR I:=1 TO N DO B (I) :=A (I) ;
WRITELN(CR,'### HEAP SORT ###');
WRITELN(CR);
HEAPSORT(B);
WRITELN('3');
OUT(T1,T2,B);
FOR I:=1 TO N DO B (I) :=A (I) ;
WRITELN(CR,'### QUICK SORT ###');
WRITELN(CR);
QUICKSORT(B);
WRITELN('4');
OUT(T1,T2,B);
WRITELN(CR,'### MERGE SORT ###');
WRITELN(CR);
MERGESORT(N A);
WRITELN('5');
IF ODD(SEED)
  THEN OUT(T1,T2,A)
  ELSE OUT(T1,T2,B);
CLOSE(CR)

```

END.

OUTPUT:

(• TOTAL: DAT •)

N=1000

INSERT SORT

8

22

87

142

149

230

237

```

247
280
288
321
339
358
418
437
TIME=4.252010E+01 SECOND
### BINARY SORT ###
8
.
.
.
437
TIME=3.436017E+01 SECOND
### HEAP SORT ###
8
.
.
.
437
TIME=3.840065E+00 SECOND
### QUICK SORT ###
8
.
.
.
437
TIME=2.239752E+00 SECOND
### MERGE SORT ###
8
.
.
.
437
TIME=3.219938E+00 SECOND
FIG TOTAL3 PROGRAM

```

第四个程序 (TOTAL4.PAS)

功能：根据给定的迷宫图，计算机探测前进方位，判断正确与否，找出迷宫路径。

老鼠走迷宫的试验，是实验心理学中的一个古典问题。用计算机解迷宫路径的程序，就是仿照老鼠走迷宫而设计的，也是对盲人走路的一个机械模仿。

数据文件 MAZE.DAT 是给定的用二维数组表示的迷宫图。迷宫通路上的坐标点为 0，

其余坐标点皆为 1。

MOVE.DAT 为表示探测方位的数据文件，它有八个方位：

1. 正北； 2. 东北；
3. 正东； 4. 东南；
5. 正南； 6. 西南；
7. 正西； 8. 西北。

例如：(-1, 0)，表示行要减 1，列不变，代表探测正北方向。用 1 表示。(0, 1) 表示行不变，列增加 1，表示探测正东方向。用 3 代表这个方位。

程序中 (M, N) 代表迷宫方阵的大小，(I, J) 表示老鼠所在方位，(G, H) 表示老鼠下一步方位，堆栈 STACK 记忆走过的路径，MARK 为标志数组。

具体步骤如下：

1. 令计算机进入迷宫入口；
2. 在当前位置上向八个方位（从“正北”方位起，顺时针探测直到“西北”方位）探测前进方位；
3. 向探测到的通路方位（即用 0 表示的通路）迈进一步，并记下迈进之前的方位；
4. 重复 2、3；
5. 若找不到前进通路（如七个方位皆为 1）只能后退一步。再重复 2、3，寻找另一条新的路径。
6. 重复 2—5，直到迷宫的“出口”，或判断后宣布：这是一个不存在通路的死迷宫。

为了避免检测边界状态，把迷宫 MAZE (1..M, 1..N) 扩大为 MAZE (0..M+1, 0..N+1)，且令 0 行 0 列，M+1 行和 N+1 列（即四周一圈的坐标点均为 1）作为边界，这样迷宫路径始终在边界之内。

源程序、数据文件和输出结果详见 FIG TOTAL4 PROGRAM。

```
PROGRAM TOTAL4 (INPUT, OUTPUT),
CONST
  M=9; N=11;
VAR
  MAZE, MARK: ARRAY (0..M, 0..N) OF INTEGER;
  MOVE: ARRAY (1..8, 1..2) OF INTEGER;
  STACK: ARRAY (0..200, 1..3) OF INTEGER;
  I, J, G, H, K, V, TOP: INTEGER;
  C, D: BOOLEAN;
  F, Q, P: TEXT;
BEGIN
  RESET(P, 'MAZE', 'DAT');
  REWRITE(F, 'TOTAL4', 'DAT'); WRITELN(F);
  FOR I:=0 TO M DO
    FOR J:=0 TO N DO
      READ(P, MAZE (I, J));
  FOR I:=0 TO M DO
    FOR J:=0 TO N DO MARK (I, J) := 0;
```

```

RESET(Q, 'MOVE', 'DAT'),
FOR I:=1 TO 8 DO
READ(Q, MOVE (I, 1) , MOVE (I, 2) );
I:=1; J:=1; V:=1; MARK (1, 1) :=1; TOP:=0;
REPEAT
G:=I+MOVE (V, 1) ; H:=J+MOVE (V, 2) ;
IF (MAZE (G, H) =0) AND (MARK (G, H) =0)
THEN
BEGIN
MARK (G, H) :=1; TOP:=TOP+1;
STACK (TOP, 1) :=I; STACK (TOP, 2) :=J;
STACK (TOP, 3) :=V;
I:=G; J:=H; V:=1
END
ELSE
IF V<8
THEN V:=V+1
ELSE
BEGIN
WHILE (TOP>0) AND (STACK (TOP, 3) =8) DO
TOP:=TOP-1;
IF TOP>0
THEN
BEGIN
I:=STACK (TOP, 1) ;
J:=STACK (TOP, 2) ;
V:=STACK (TOP, 3) +1;
TOP:=TOP-1
END
END;
C:=(G=M-1) AND (H=N-1); D:=(TOP=0) AND (V=4);
UNTIL (C AND (MAZE (G, H) =0)) OR D;
IF TOP=0
THEN WRITELN(F, 'NO PATH HAS BEEN FOUND')
ELSE
BEGIN
TOP:=TOP+1; STACK (TOP, 1) :=G; STACK (TOP, 2) :=H;
STACK (TOP, 3) :=0;
WRITELN(F, '10, YOUR MAZE HAS PATH AS BELOW');
WRITELN(F);
K:=1;
REPEAT
WRITE(F, '(' , STACK (K, 1) :2, STACK (K, 2) :3, STACK (K, 3) :3, ')');
IF (K MOD 4)=0

```

```

        THEN WRITELN(F);
        K:=K+1;
    UNTIL K>TOP;
    WRITELN(F)
END;
K:=1;
REPEAT
    I:=STACK (K, 1) , J:=STACK (K, 2) , MAZE (I, J) :=4;
    K:=K+1;
UNTIL K>TOP;
K:=1;
FOR I:=0 TO M DO
FOR J:=0 TO N DO
    BEGIN
        WRITE(F, MAZE (I, J) :2);
        IF (K MOD 12)=0
            THEN
                WRITELN(F), K:=K+1
    END;
WRITELN(F, 'Note, The char "4" represents path of MAZE ');
CLOSE(F)
END.

```

(• MAZE.DAT •)

```

1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 0 1 0 0 1 1
1 0 0 0 0 0 0 1 1 0 1 1
1 0 1 1 1 1 1 1 1 0 0 1
1 1 0 0 0 0 1 1 1 1 1 1
1 1 1 0 1 0 0 0 0 0 1 1
1 1 1 0 1 0 0 1 1 0 1 1
1 1 1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1

```

(• MOVE.DAT •)

```

-1      0
-1      1
0      1
1      1
1      0
1      -1
0      -1
-1      -1

```

OUTPUT:

(• TOTAL.DAT •)

YOUR MAZE HAS PATH AS BELOW

```
( 1 1 3 ) ( 1 2 3 ) ( 1 3 3 ) ( 1 4 3 )
( 1 5 3 ) ( 1 6 5 ) ( 2 6 5 ) ( 3 6 7 )
( 3 5 7 ) ( 3 4 7 ) ( 3 3 7 ) ( 3 2 6 )
( 4 1 4 ) ( 5 2 3 ) ( 5 3 3 ) ( 5 4 3 )
( 5 5 4 ) ( 6 6 3 ) ( 6 7 3 ) ( 6 8 3 )
( 6 9 5 ) ( 7 9 4 ) ( 8 10 0 )
```

```
1 1 1 1 1 1 1 1 1 1 1 1
1 4 4 4 4 4 4 1 1 1 1 1
1 1 1 1 1 1 4 1 0 0 1 1
1 0 4 4 4 4 4 1 1 0 1 1
1 4 1 1 1 1 1 1 1 0 0 1
1 1 4 4 4 4 1 1 1 1 1 1
1 1 1 0 1 0 4 4 4 4 1 1
1 1 1 0 1 0 0 1 1 4 1 1
1 1 1 1 0 0 0 1 1 0 4 1
1 1 1 1 1 1 1 1 1 1 1 1
```

Note: The char '4' represents path of MAZE

FIG TOTAL4 PROGRAM

第五个程序 (TOTAL5.PAS)

功能：用最优化方法计算液体混合物导热系数的新关联式的经验常数，及其导热系数平均绝对误差的统计。

“液体混合物的导热系数”这个课题，已经进行了数十年的实验和理论研究，在此基础上提出的若干有关导热系数的预计方法，在工程计算中得到了初步应用。清华大学化学化工系童景山教授等提出的液体混合物导热系数的新关联式，使计算公式具有明确的物理意义，更具有普遍性（即不仅适用于双元素，也适用于多元素体系），也使计算的精确性和可靠性有了提高。

对双元素来说，液体混合物导热系数新关联式的计算公式有两种：

$$\lambda_m = \phi_1^2 \lambda_1 + \phi_2^2 \lambda_2 + \phi_1 \phi_2 (\lambda_1 + \lambda_2) e^{-\alpha(1-B_{12})} \quad \dots\dots (1)$$

$$\lambda_m = \phi_1^2 \lambda_1 + \phi_2^2 \lambda_2 + \phi_1 \phi_2 (\lambda_1 + \lambda_2) e^{-\gamma \left(\frac{|\lambda_1 - \lambda_2|}{\lambda_1 + \lambda_2} \right)^2} \quad \dots\dots (2)$$

其中 λ_m 、 λ_1 、 λ_2 为混合物、组分 1 和组分 2 的导热系数，由实验测出其数据。

ϕ_1 、 ϕ_2 为组分 1 和组分 2 的有效容积分数，根据组分 1 和组分 2 的重量百分比 G_1 、 G_2 ，以及临界容积 V_{c1} 和 V_{c2} 可以计算出来。

B_{12} 为组分 1 和组分 2 的导热系数比值，它要求其值小于 1。

对于上述公式，本程序对 89 组双元系的液体混合液共 449 个（组）数据点，采用最优化方法——即最小二乘法建立目标函数，用 0.618 方法在计算机上进行运算，求出经验常数 α 和 γ 的值， α 为 0.447464， γ 为 0.447289。

根据 α 和 γ 两个经验常数的数值，进一步计算各组数据点的误差及其绝对平均误差，最后求出八十九组双元系液合液的总绝对平均误差。从最终结果看到，用 α 方法求出八十九

组双元系的总绝对平均误差是2.65%，用 γ 方法则为3.0%。这表明用新关联式求出的液体混合物导热系数的预计方法是可采用的有重要意义的。

实现的源程序、输入的实验数据文件和输出结果的示意文件如FIG TOTAL5 PROGRAM所示。

```

PROGRAM TOTAL5(INPUT, OUTPUT);
  (* HEAT TRANSFER COEFFICIENT OF LIQUID MIXTURE *)
CONST
  N=449; C=0.618
TYPE
  ARIN=ARRAY (1..N) OF REAL;
  ARCH=PACKED ARRAY (1..40) OF CHAR;
VAR
  H1, H2, H0:TEXT;
  P :CHAR;
  CHD:ARRAY (1..N, 1..8) OF REAL;
  M :ARRAY (1..N) OF INTEGER;
  Ag, Bg, ERROR1, ERROR2:ARRAY (1..N) OF REAL;
  L, I, J, T1, T2, No, Io, INTEGER;
  S, S1, S2, S3, r, a, E0, E, E1, E2, E3, E4;
  Fe, F, F0, F1, F3, F4, Q, Q1, Q2, Q3, B, K0:REAL;
  ARCH1:ARCH;
PROCEDURE TJ(N1, N2:INTEGER);
  BEGIN
    REWRITE(H0, 'LP:');
    FOR I:=N1 TO N2 DO
      BEGIN
        M (I) :=1;
        IF CHD (I, 5) <=CHD (I, 6)
          THEN B:=CHD (I, 5) /CHD (I, 6)
          ELSE B:=CHD (I, 6) /CHD (I, 5);
        Q:=CHD (I, 3) /CHD (I, 7) +CHD (I, 4) /CHD (I, 8);
        Q1:=(CHD (I, 3) /CHD (I, 7))/Q;
        Q2:=(CHD (I, 4) /CHD (I, 8))/Q;
        K0:=Q1 * Q1 * CHD (I, 5) +Q2 * Q2 * CHD (I, 6);
        S3:=-2 * r * ABS(CHD (I, 5) -CHD (I, 6))/(CHD (I, 5) +CHD (I, 6));
        Ag (I) :=K0+(CHD (I, 5) +CHD (I, 6)) * Q1 * Q2 * EXP(-S * (1-B));
        Bg (I) :=K0+(CHD (I, 5) +CHD (I, 6)) * Q1 * Q2 * EXP (S3);
        ERROR1 (I) :=(CHD (I, 2) --Ag (I))/CHD (I, 2);
        ERROR2 (I) :=(CHD (I, 2) --Bg (I))/CHD (I, 2);
      END;
    WRITE(H0, ' ':10, ' * HEAT TRANSFER COEFFICIENT STATISTICAL');
    WRITELN(H0, 'TABLE FOR LIQUID MIXTURE *');
    WRITELN(H0);
  END;

```

```

WRITELN(H0, ' ', 'NAME OF LIQUID MIXTURE:', ARCH1);
WRITELN(H0);
WRITE(H0, ' 1 2 3 4 5 6 ');
WRITELN(H0, ' 7 8 9 10 11 12');
WRITE(H0, ' XH Am G1 G2 A1 A2 ');
WRITELN(H0, ' 1/Vc1 1/Vc2 Ag Error1 Bg Error2');
WRITELN(H0);
FOR I:=N1 TO N2 DO
  BEGIN
    WRITE(H0, M (I) :3, ' ');
    FOR J:=2 TO 8 DO
      BEGIN
        CASE J OF
          2: WRITE(H0, CHD (M (I) , J) :5:1, ' ');
          3: WRITE(H0, CHD (M (I) , J) :4:1, ' ');
          4: WRITE(H0, CHD (M (I) , J) :4:1, ' ');
          5: WRITE(H0, CHD (M (I) , J) :5:1, ' ');
          6: WRITE(H0, CHD (M (I) , J) :5:1, ' ');
          7: WRITE(H0, CHD (M (I) , J) :5:3, ' ');
          8: WRITE(H0, CHD (M (I) , J) :5:3, ' ');
        END
      END;
    WRITE(H0, Ag (I) :7:3, ERROR1 (I) :8:4);
    WRITELN(H0, Bg (I) :7:3, ERROR2 (I) :8:4);
  END;
S1:=0; S2:=0;
FOR I:=N1 TO N2 DO
  BEGIN
    S1:=S1+ABS(ERROR1 (I) );
    S2:=S2+ABS(ERROR2 (I) );
  END;
WRITE(H0, ' ', :39, 'ABS AVERAGE ERROR: ');
WRITELN(H0, (S1/(N2-N1+1)):8:4, ' ', :8, S2/(N2-N1+1):8:4);
WRITELN(H0);
WRITELN(H0);
CLOSE(H0)
END;
PROCEDURE OFe(VAR E:REAL);
BEGIN
  Fe:=0;
  FOR I:=1 TO N DO
    BEGIN
      Q:=(CHD (I,3) /CHD (I,7) +CHD (I,4) /CHD (I,8) );
      Q2:=(CHD (I,4) /CHD (I,8) )/Q;
    END
  END;

```



```

      Q1:=CHD (I,3) /CHD (I,7) )/Q,
      IF CHD (I, 5) <=CHD (I, 6)
      THEN B:=CHD (I, 5) /CHD (I, 6)
      ELSE B:=CHD (I, 6) /CHD (I, 5)
      F0:=CHD (I,2) -Q1 * Q1 * CHD (I,5) -Q2 * Q2 * CHD (I,6)
      Q3:=Q1 * Q2 * (CHD (I,5) +CHD (I,6) );
      S3:=-2 * E * ABS(CHD (I,5) -CHD (I,6) )/(CHD (I,5) +CHD (I,6) );
      IF L=1
      THEN F1:=Q3 * EXP (-E * (1-B))
      ELSE F1:=Q3 * EXP (S3);
      F:=(F0-F1) * (F0-F1);
      Fe:=Fe+F
    END
  END)
BEGIN      (* MAIN PROGRAM *)
  P:='N',
  RESET(H1,'CHD');
  REWRITE(H2,'LP:');
  FOR I:=1 TO N DO
  FOR J:=1 TO 8 DO
    BEGIN
      READ(H1, CHD (I, J) );
      CASE J OF
        1:WRITE(CHD (I, J) :4:1,' ');
        2:WRITE(CHD (I, J) :5:1,' ');
        3:WRITE(CHD (I, J) :4:1,' ');
        4:WRITE(CHD (I, J) :4:1,' ');
        5:WRITE(CHD (I, J) :5:1,' ');
        6:WRITE(CHD (I, J) :5:1,' ');
        7:WRITE(CHD (I, J) :5:3,' ');
        8:WRITE(CHD (I, J) :5:3,' ');
      END;
      IF J=8
      THEN WRITELN;
      END;
      E1:=0.0, E2:=1.0, E0:=0.001;
      N0:=TRUNC(LN(E0/(E2-E1))/(-0.4812))+2;
      I0:=N0;
      FOR L:=1 TO 2 DO
        BEGIN
          E4:=E1+C * (E2-E1);
          QFe(E4);
          F4:=Fe;
          E3:=E2-C * (E2-E1);

```

```

QFe(E3);
F3:=Fe;
WHILE I0>1 DO
  BEGIN
    I0:=I0-1;
    IF F3<=F4
    THEN
      BEGIN
        E2:=E4; E4:=E3; F4:=F3;
        E3:=E2-C*(E2-E1);
        QFe(E3);
        F3:=Fe
      END
    ELSE
      BEGIN
        E1:=E3; E3:=E4; F3:=F4;
        E4:=E1+C*(E2-E1);
        QFe(E4);
        F4:=Fe
      END
    END.
    IF F3<=F4
    THEN
      BEGIN
        IF L=1
        THEN
          BEGIN
            a:=E3;
            WRITELN(H2,' a=',E3:8:6,' ':18,'F=',F3:8:6);
            WRITE(H2,' (E4-E3)/E3=',(E4-E3)/E3:8:6);
            WRITELN(H2,' ':8,'(F4-F3)/F3=',(F4-F3)/F3:8:6)
          END
        ELSE
          BEGIN
            r:=E3;
            WRITEN(H2,' r=',E3:8:6,' ':18,'F=',F3:8:6);
            WRITE(H2,' (E4-E3)/E3=',(E4-E3)/E3:8:6);
            WRITELN(H2,' ':8,'(F4-F3)/F3=',(F4-F3)/F3:8:6)
          END
        END
      ELSE
        BEGIN
          IF L=1
          THEN

```

```

      BEGIN
        a:=E4;
        WRITELN(N2,' a=',E4:8:6,' ':18,'F=',F4:8:6);
        WRITE(H2,' (E4-E3)/E4=',(E4-E3)/E4:8:6);
        WRITELN(H2,' ':8,'(F3-F4)/F4=',(F3-F4)/F4:8:6)
      END
    ELSE
      BEGIN
        r:=E4;
        WRITELN (H2,'r=', E4:8:6,' ':18,'F=',F4:8:6);
        WRITE(H2,' (E4-E3)/E4=',(E4-E3)/E4:8:6);
        WRITELN(H2,' ':8,'(F3-F4)/F4=',(F3-F4)/F4:8:6)
      END
    END
  END
REPEAT
  WRITELN('PLEASE INPUT NAME OF LIQUID MIXTURE '),
  READLN(ARCH1);
  WRITELN('PLEASE INPUT N1=?'), READLN(T1);
  WRITELN('PLEASE INPUT N2=?'), READLN(T2);
  TJ(T1, T2);
  WRITELN('FINISHED ?'), READLN(P);
  UNTIL P='Y';
  S1:=0;
  FOR I:=1 TO N DO S1:=S1+ABS(ERROR1 (I) );
  WRITELN(H2,' ':8,'TOTAL AVERAGE ABS ERROR(S1):',S1/N:8:4);
  S2:=0;
  FOR I:=1 TO N DO S2:=S2+ABS(ERROR2 (I) );
  WRITELN(H2,' ':8,'TOTAL AVERAGE ABS ERROR(S2):',S2/N:8:4);
  CLOSE(H2)
END.

```

```

      ( *      CHD.DAT      * )
001  147.9  20.0  80.0  200.5  140.0  0.262  0.291
002  149.5  25.0  75.0  200.5  140.0  0.262  0.291
003  162.2  40.0  60.0  200.5  140.0  0.262  0.291
004  164.2  50.0  50.0  200.5  140.0  0.262  0.291
005  168.2  60.0  40.0  200.5  140.0  0.262  0.291
006  188.0  80.0  20.0  200.5  140.0  0.262  0.291
007  143.9  10.0  90.0  166.2  141.9  0.351  0.302
.....
446  171.0  91.1  08.9  160.0  629.0  0.311  0.322
447  215.0  77.8  22.2  160.0  629.0  0.311  0.322
448  318.0  50.6  49.2  160.0  629.0  0.311  0.322
449  382.0  35.4  64.6  160.0  629.0  0.311  0.322

```

OUTPUT:

• HEAT TRANSFER COEFFICIENT STATISTICAL TABLE FOR LIQUID MIXTURE •
NAME OF LIQUID MIXTURE:1.ACETALDEHYDE-TOLUENE,T=0

1	2	3	4	5	6	7	8	9	10	11	12
XH	Am	G1	G2	A1	A2	1/Vc1	1/Vc2	Ag	Error1	Bg	Error2
1	147.9	20.0	80.0	200.5	140.0	0.262	0.291	145.833	0.0140	144.637	0.0221
2	149.5	25.0	75.0	200.5	140.0	0.262	0.291	147.866	0.0109	146.479	0.0202
3	162.2	40.0	60.0	200.5	140.0	0.262	0.291	155.227	0.0430	153.507	0.0536
4	164.2	50.0	50.0	200.5	140.0	0.262	0.291	161.114	0.0188	159.361	0.0290
5	168.2	60.0	40.0	200.5	140.0	0.262	0.291	167.727	0.0028	166.078	0.0126
6	188.0	80.0	20.0	200.5	140.0	0.262	0.291	182.935	0.0269	181.880	0.0326
ABS AVERAGE ERROR									0.0194	0.0284	

.....

• HEAT TRANSFER COEFFICIENT STATISTICAL TABLE FOR LIQUID MIXTURE •
NAME OF LIQUID MIXTURE:89.PYRIDINE-WATER,T=40

1	2	3	4	5	6	7	8	9	10	11	12
XH	Am	G1	G2	A1	A2	1/Vc1	1/Vc2	Ag	Error1	Bg	Error2
447	215.0	77.8	22.2	160.0	629.0	0.311	0.322	223.418	-0.0392	200.214	0.0404
448	318.0	50.6	49.2	160.0	629.0	0.311	0.322	331.235	-0.0416	305.896	0.0382
449	382.0	35.4	64.6	160.0	629.0	0.311	0.322	407.533	-0.0668	384.074	-0.0054
ABS AVERAGE ERROR:									0.0492	0.0282	

r=0.447464

(E4-E3)/E4=0.000390

r=0.447289

(E4-E3)/E3=0.000381

F=18158.640000

(F3-F4)/F4=0.000001

F=46943.040000

(F4-F3)/F3=0.001297

TOTAL AVERAGE ABS ERROR(S1) : 0.0265

TOTAL AVERAGE ABS ERROR(S2) : 0.0300

FIG TOTAL5 PROGRAM

第六个程序 (TOTAL6.PAS)

功能：对某市各高等工科院校的教师、学生和研究生进行统计，求出各院校的学生对教师，研究生对导师以及教师队伍中各种成分的比例。

已知现有两个数据文件：一个是各类人员数据文件 XU1.DAT，另一个是院校名称的文件 NAME.DAT。这两个文件存在着——对应的关系，为简化起见，前一个数据文件的数据库类型是整数，后一个文件由字符构成。要求将对两个文件进行处理所得的统计汇总表，送入另一个数据文件 (TOTAL6.DAT) 中去，并打印出来。

程序中，把各类人员的统计运算及汇总表格模式，放在命名为 TJ 的过程之中实现，这样主程序一目了然，简单明了。

读者可能会问，这儿统计才 15 个单位，何必还要用过程呢？如果只是一次统计 15 个单位，不用过程，自然也是可以的。但作为一个有一定实用价值的应用例题来看，本程序这样处理是恰当的。如果统计的单位不是 15 个，而是几百，几千个，而且是多次地进行统计，那末本程序把统计部分单独作为一个过程就很必要。

另外，本程序仅仅处理两个数据文件，如果处理的数据文件有多个，比如说七八个，十几个，输出数据文件也有好几个，那该怎么办呢？应该说方法是类似的，模仿本程序的做法，便可实现。

FIG TOTAL6 PROGRAM 列出了源程序，二个输入数据文件，一个输出统计汇总表文件。

```
PROGRAM TOTAL6(INPUT OUTPUT);
(* NUMBER STATISTICAL TABLE FOR COLLEGES
   AND UNIVERSITIES, JUN-81 *)
TYPE
  ARCH=PACKED ARRAY (1..15) OF CHAR;
CONST
  N=115;
VAR
  I, J, K, N1, N2, A1, A2, S2, S3 :INTEGER;
  S4, S5, S6, S7, S8, S9, S10, T11 :INTEGER;
  A12, B13, C14, D15, E16 :REAL;
  CH :CHAR;
  A, B, C, D, E:ARRAY (101..N) OF REAL;
  T :ARRAY (101..N) OF INTEGER;
  NAME :ARRAY (101..N) OF ARCH;
  XU1 :ARRAY (101..N,1..10) OF INTEGER;
  M1, M2, M :TEXT;
PROCEDURE TJ(N1, N2:INTEGER);
BEGIN
  REWRITE(M, 'TT:');
  FOR I:=N1 TO N2 DO
    FOR J:=1 TO 10 DO
```

```

BEGIN
  READ(M1, XU1 (I, J) );
  WRITE(XU1 (I, J) :5, ' ');
  IF J=10
    THEN WRITELN
  END;
FOR I:=N1 TO N2 DO
  BEGIN
    READLN(M2, NAME (I) );
    WRITELN(NAME (I) );
  END;
FOR I:=N1 TO N2 DO
  BEGIN
    T (I) :=XU1 (I,2) +XU1 (I,3) +XU1 (I,4) +XU1 (I,5)
              +XU1 (I,6) +XU1 (I,7) ;
    A (I) :=XU1 (I,9) /T (I) ;
    B (I) :=XU1 (I,10) /(XU1 (I,2) +XU1 (I,3) );
    C (I) :=(XU1 (I,2) +XU1 (I,3) )/T (I) ;
    D (I) :=XU1 (I,4) /T (I) ;
    E (I) :=XU1 (I,6) /T (I)
  END;
WRITE(M, ' ':10, ' * COLLEGES AND UNIVERSITIES NUMBER
      STATISTICAL');
WRITELN(M, 'TABLE OF TEACHERS, STUDENTS AND GRADUA-
      TES * ');
WRITELN(M);
WRITE(M, 'ORD COLLEGE-NAME PROF A,PROF LECT TECH ');
WRITE(M, 'ASSI OTHE E,Y S S G ');
WRITELN(M, ' T S/T G/2+3 2+3/T 4/T 6/T ');
WRITELN(M);
WRITE(M, ' No NAME T1 T2 T3 ');
WRITE(M, ' T4 T5 T6 UN U G ');
WRITE(M, 'T A B C D E ');
WRITELN(M);
FOR I:=N1 TO N2 DO
  BEGIN
    WRITE(M, I:3, ' ');
    WRITE(M, NAME (I) );
    FOR J:=2 TO 10 DO WRITE(M, XU1 (I,J) :3, ' ':4);
    WRITE(M, T (I) :4, ' ', A (I) :4:1, ' ', B (I) :4:1, ' ');
    WRITELN(M, C (I) :4:2, ' ', D (I) :4:2, ' ', E (I) :4:2, ' ');
  END;
WRITELN(M)
S2:=0, S3:=0, S4:=0, S5:=0, S6:=0, S7:=0, S8:=0, S9:=0, S10:=0;

```

1

```

FOR I:=101 TO N DO
  BEGIN
    S2:=S2+XU1 (I,2) ; S3:=S3+XU1 (I,3) ; S4:=S4+XU1 (I,4) ;
    S5:=S5+XU1 (I,5) ; S6:=S6+XU1 (I,6) ; S7:=S7+XU1 (I,7) ;
    S8:=S8+XU1 (I,8) ; S9:=S9+XU1 (I,9) ; S10:=S10+XU1 (I,10)
  END;
  T11:=S2+S3+S4+S5+S6+S7;
  A12:=S9/T11; B13:=S10/(S2+S3); C14:=(S2+S3)/T11;
  D15:=S4/T11; E16:=S6/T11;
  WRITE(M,' TOTAL..... ',S2:3,' ':4, S3:3,' ':3);
  WRITE(M,S4:4,' ':4,S5:3,' ':3, S6:4,' ':4, S7:3,' ':3, S8:4,' ':2);
  WRITE(M,S9:6,' ':3,S10:4,' ':4, T11:4,' ':3,A12:4:1,' ':3,B13:4:1);
  WRITELN(M,' ':3, C14:4:2,' ':3, D15:4:2,' ':3,E16:4:2,' ':3);
  CLOSE(M);
END;
BEGIN      (• MAIN PROGRAM •)
  CH:='N';
  RESET(M1, 'XU1');
  RESET(M2, 'NAME');
  REPEAT
    WRITELN('PLEASE INPUT NUMBER N1=? '); READLN(A1);
    WRITELN('PLEASE INPUT NUMBER N2=? '); READLN(A2);
    TJ(A1, A2);
    WRITELN('FINISHED? (YES, INPUT 'Y', ELSE INPUT 'N')');
    READLN(CH)
  UNTIL CH='Y'
END

```

```

      (• XU1.DAT •)
101 08 29 048 04 030 01 120 424 33
102 15 27 067 04 061 10 210 811 58
103 10 29 086 02 066 14 170 667 38
104 04 44 117 07 094 13 160 629 64
105 06 28 064 04 053 13 230 862 69
106 09 31 092 01 078 16 250 923 35
107 11 32 065 00 083 09 220 816 40
108 04 23 073 02 090 12 190 747 31
109 03 15 060 01 046 08 125 494 73
110 01 12 095 09 081 25 195 763 53
111 01 27 097 01 110 17 250 977 44
112 02 20 081 05 071 07 225 895 57
113 04 29 144 04 099 16 250 976 47
114 03 06 041 12 047 00 085 331 26
115 18 81 097 01 128 23 130 504 43

```

```

      (• NAME.DAT •)
Architecture
Civil-Environ
Hydraulic
Mechanical
Instrument
Thermal
Electrical
Radio
Computer
Automation
Engin-Physics
Chemical
Mechanics
Economic
Fundamental

```

OUTPUT: (• TT: •)

• COLLEGES AND UNIVERSITIES NUMBER STATISTICAL TABLE OF TEACHERS, STUDENTS AND GRADUATES •

ORD	COLLEGE-NAME PROF A. PROF LECT TECH ASSI OTH E V S S																	G	I	S/I			G/2+3			2+3/T	4/I	5/T
NO	NAME	T1	T2	T3	T4	T5	T6	UN	U	G	T	A	B	C	D	E												
101	Architecture	8	29	48	4	30	1	120	424	33	120	3.5	0.9	0.31	0.40	0.20												
102	Civil-Environ	15	27	67	4	61	10	210	311	58	184	4.4	1.4	0.23	0.36	0.33												
103	Hydraulic	10	29	86	2	60	14	170	667	38	207	3.2	1.0	0.19	0.42	0.31												
104	Mechanical	4	44	117	7	94	13	160	629	64	272	2.3	1.3	0.17	0.42	0.30												
105	Instrument	6	28	64	4	53	13	239	862	69	158	5.1	2.0	0.20	0.38	0.32												
106	Thermal	9	31	92	1	78	16	250	923	30	227	4.1	0.3	0.18	0.41	0.34												
107	Electrical	11	32	65	0	83	9	220	816	40	200	4.1	0.9	0.22	0.33	0.42												
108	Radio	4	23	73	2	90	12	190	747	37	204	3.7	1.1	0.13	0.30	0.44												
109	Computer	3	15	60	1	46	8	125	494	73	133	3.7	4.1	0.14	0.45	0.35												
110	Automation	1	12	95	9	81	25	195	763	53	223	3.4	4.1	0.06	0.43	0.30												
111	Engin-Physics	1	27	97	1	110	17	250	977	14	253	3.9	1.6	0.11	0.38	0.43												
112	Chemical	2	20	81	5	71	7	225	895	67	186	4.8	2.6	0.12	0.44	0.38												
113	Mechanics	4	29	144	4	99	16	250	976	47	296	3.3	1.4	0.11	0.49	0.33												
114	Economic	3	6	41	12	47	0	85	331	26	109	3.0	2.9	0.08	0.38	0.43												
115	Fundamental	18	81	97	1	128	23	130	504	43	343	1.4	0.4	0.28	0.28	0.37												
TOTAL.....		99	433	1227	57	1137	194	2810	10819	711	3137	3.4	1.3	0.17	0.39	0.36												

FIG TOTAL6 PROGRAM

第七个程序 (TOTAL7.PAS)

本程序的功能，基本上与第六个程序相同。但就其处理方式来说，两者差别很大。

在第六个程序中，处理的数据文件可以有很多个，但每一个数据文件只能是同一类型的数据（或是整数，或是实数，或是字符等等），它们单独存放在用户盘中。

但在日常生活中，尤其是作人员和物资统计，人员或物资的情况既有整数，又有实数，又有字符，甚至还有别的，这怎么办呢？如果把这些基本情况组成的一个统一的数据文件，根据数据的类型，拆成几个数据文件，不仅很麻烦，有时甚至是很困难的。即使能拆成几个数据文件，也因为它们失去了明确的意义而使这些文件的组成和阅读增加了困难。

对于各种管理统计工作，经常遇到要组成并处理包含各种数据类型的数据文件。本程序正是为解决这方面问题而设置的，它处理下面所示的 XUNAME.DAT 这样的数据文件：

(• XUNAME.DAT •)										
101	Architecture	08	29	043	04	030	01	120	424	33
102	Civil-Environ	15	27	061	04	061	10	210	811	58
103	Hydraulic	10	21	086	02	066	14	170	661	38
104	Mechanical	04	44	117	07	094	13	160	629	64
105	Instrument	06	28	001	04	053	13	230	862	69
106	Thermal	09	31	092	01	078	16	250	923	35
107	Electrical	11	32	005	00	083	09	220	816	40
108	Radio	04	23	073	02	090	12	190	747	31
109	Computer	03	15	060	01	046	08	125	494	73
110	Automation	01	12	095	09	081	25	195	763	53
111	Engin-Physics	01	27	097	01	110	17	250	977	44
112	Chemical	02	20	081	05	071	07	225	895	57
113	Mechanics	04	29	144	04	099	16	250	976	47
114	Economic	03	06	041	12	047	06	085	331	26
115	Fundamental	18	81	097	01	128	23	130	504	43

本程序对记录类型的数据文件的处理方式有一个特点，那就是必须对全部数据文件进行一次处理，而不能象第六个程序那样，对数据文件进行多次分段处理。

对数据文件的组成和处理方式，是采用本程序的方式，还是采用第六个程序所示的方式，要根据具体情况而定。

有关本程序的内容和处理结果的输出文件见 FIG TOTAL7 PROGRAM.

```

PROGRAM TOTAL7(INPUT,OUTPUT);
TYPE
  ARCH=PACKED ARRAY (1..15) OF CHAR;
  QR=RECORD
    NAME:ARCH;
    No, T1, T2, T3, T4, T5, T6, UN, U, G:INTEGER;
  END;
VAR
  I,J,X,S2,S3,S4,S5,S6,S7,S8,S9,S10,T11:INTEGER;
  A12,B13,C14,D15,E16:REAL;

```

```

CH :CHAR,
M1,M :TEXT,
P :QH,
BEGIN
  REWRITE(M,'TOTAL7','DAT');
  RESET(M1,'XUNAME','DAT');
  WRITE(M,' :10. ' * COLLEGES AND UNIVERSITIES NUMBER STA-
    TISTICAL');
  WRITELN(M,'TABLE OF TEACHERS, STUDENTS AND GRADUATES *');
  WRITELN(M);
  WRITE(M,'ORD COLLEGE-NAME PROF A.PROF LECT TECH ');
  WRITE(M,'ASSI OTHE E.Y.S S G');
  WRITELN(M,' T S/T G/2+3 2+3/T 4/T 6/T ');
  WRITELN(M);
  WRITE(M,' No NAME T1 T2 T3 ');
  WRITE(M,' T4 T5 T6 UN U G ');
  WRITELN(M,' T A B C D E ');
  WRITELN(M);
  S2:=0; S3:=0; S4:=0; S5:=0; S6:=0; S7:=0; S8:=0; S9:=0; S10:=0;
  WHILE NOT EOF(M1) DO WITH P DO
    BEGIN
      READ(M1,No,NAME,T1,T2,T3,T4,T5,T6,UN,U,G);
      S2:=S2+T1; S3:=S3+T2; S4:=S4+T3; S5:=S5+T4; S6:=S6+T5;
      S7:=S7+T6; S8:=S8+UN; S9:=S9+U; S10:=S10+G;
      WRITE(M,NQ:3,' ',NAME,T1:3,' ':4,T2:3,' ':4,T3:3);
      WRITE(M,' ':4,T4:3,' ':4,T5:3,' ':4,T6:3,' ':4,UN:3);
      WRITE(M,' ':4,U:3,' ':4,G:3,' ':4);
      WRITE(M,' (T1+T2+T3+T4+T5+T6):4,' ':3);
      WRITE(M,' U/(T1+T2+T3+T4+T5+T6):4:1,' ':3);
      WRITE(M,' G/(T1+T2):4:1,' ':3);
      WRITE(M,' (T1+T2)/(T1+T2+T3+T4+T5+T6):4:2,' ':3);
      WRITE(M,' T3/(T1+T2+T3+T4+T5+T6):4:2,' ':3);
      WRITE(M,' T5/(T1+T2+T3+T4+T5+T6):4:2,' ':3);
      WRITELN(M);
    END;
  T11:=S2+S3+S4+S5+S6+S7;
  A12:=S9/T11; B13:=S10/(S2+S3); C14:=(S2+S3)/T11;
  D15:=S4/T11; E16:=S6/T11;
  WRITE(M,' TOTAL..... ', S2:3,' ':4, S3:3,' ':3);
  WRITE(M,' S4:4,' ':4, S5:3,' ':3, S6:4,' ':4, S7:3,' ':3, S8:4,' ':2);
  WRITE(M,' S9:5,' ':3, S10:4,' ':4, T11:4,' ':3, A12:4:1,' ':3, B13:4:1);
  WRITELN(M,' ':3, C14:4:2,' ':3, D15:4:2,' ':3, E16:4:2,' ':3);
  CLOSE(M);
END

```

OUTPUT:

TOTAL7.DAT																			
(*																			
* COLLEGES AND UNIVERSITIES NUMBER STATISTICAL TABLE OF TEACHERS, STUDENTS AND GRADUATES •																			
ORD	COLLEGE-NAME	PROF	A	PROF	LECT	TECH	ASSI	OTHE	E	Y	S	S	G	T	A	B	C	D	E
No	NAME	T1	T2	T3	T4	T5	T6	UN	U	G	T	A	B	C	D	E	6/T	4/T	6/T
101	Architecture	8	29	48	4	30	1	120	124	33	120	3.5	0.9	0.31	0.40	0.25			
102	Civil-Environ	15	27	67	4	61	10	210	811	58	184	4.4	1.4	0.23	0.36	0.33			
103	Hydraulic	10	29	86	2	66	14	170	467	38	207	3.2	1.0	0.19	0.42	0.32			
104	Mechanical	4	44	117	7	94	13	160	629	64	279	2.3	1.3	0.17	0.42	0.34			
105	Instrument	6	28	64	4	53	13	230	462	69	166	5.1	2.0	0.20	0.38	0.32			
106	Thermal	9	31	92	1	78	16	250	423	35	227	1.1	0.9	0.18	0.41	0.34			
107	Electrical	11	32	65	0	83	9	220	616	40	200	4.1	0.9	0.22	0.33	0.42			
108	Radio	4	23	73	2	90	12	190	747	31	204	3.7	1.1	0.13	0.36	0.44			
109	Computer	3	15	60	1	46	8	125	494	73	133	3.7	4.1	0.14	0.45	0.35			
110	Automation	1	12	95	9	81	25	195	763	53	223	3.4	4.1	0.06	0.43	0.36			
111	Engin-Physics	1	27	97	1	110	17	250	677	44	253	3.9	1.6	0.11	0.38	0.43			
112	Chemical	2	20	81	5	71	7	225	495	57	186	4.8	2.6	0.12	0.44	0.38			
113	Mechanics	4	29	144	4	99	16	250	976	47	296	3.3	1.4	0.11	0.49	0.33			
114	Economic	3	6	41	12	47	0	85	331	26	109	3.0	2.9	0.08	0.38	0.43			
115	Fundamental	18	881	97	1	128	23	130	504	43	348	1.4	0.4	0.28	0.28	0.37			
TOTAL.....		98	433	1227	57	1137	184	2810	10819	711	3137	3.4	1.3	0.17	0.39	0.36			

FIG TOTAL7 PROGRAM

从本程序处理结果的输出文件来看，它与第六个程序的输出数据文件一模一样。但是与第六个程序相比，本程序更简明扼要，程序的长度几乎缩短了一半。这是本程序的一个优点。但在某些情况下，又有不足之处，因为它不能对记录类型的数据文件进行多次分段处理。

本程序处理的 XUNAME.DAT 这个数据文件中，包含的数据类型只是字符与整数，没有更多的其它数据类型。但这并不影响到它的通用性。包括输出数据文件，也不限于只是一个。请读者参照本例题，举一反三，在实际中作进一步的发挥和应用。

第八个程序 (TOTAL8.PAS)

功能：求解 AOE 网的关键路径。

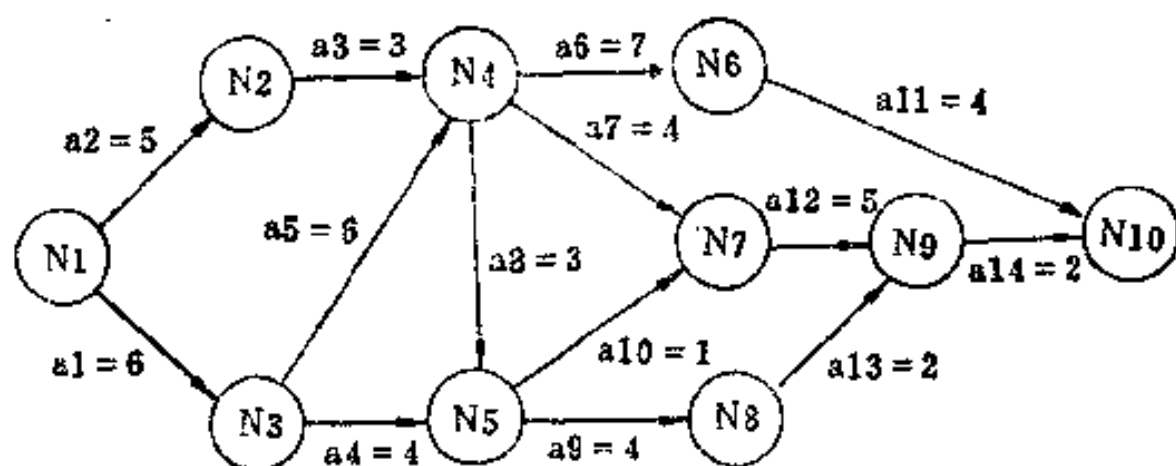


图 11.8-1 某工程的 AOE 网

图 11.8-1 为某工程的 AOE (Activity On Edge) 网。图中 N1 至 N10 表示该工程由十项子工程组成。有向边 a1 至 a14 表示各子工程的活动情况，其权代表各项子工程持续的时间。

对于 AOE 网，研究的问题是：

- (1) 完成整个工程至少需要多少时间；
- (2) 哪些活动是影响工程进度的关键。

完成工程的最少时间，是从开始点到完成点的最长路径长度（路径长度是这条路上的活动持续时间之和），人们把路径长度最长的路径称为关键路径。

分析关键路径的目的为辨别哪些是关键活动，以便提高关键活动的工效，缩短整个工程的期限。

程序中采用了十字链表的结构来表示有向图 AOE 网。把每个子工程视为一个结点，并设置五个域，当作一个记录。其中 TAIL 和 HEAD 域分别表示有向边的尾顶点 J 和头顶点 K，DUT 表示活动的持续时间，HLINK 链接以 K 为头的另一有向边，TLINK 为链接以 J 为尾的另一有向边。

在用这种结构表示的 AOE 网时，求解关键路径、邻接表和逆邻接表的程序及其结果，详见 FIG TOTAL8 PROGRAM。

从输出结果看，AOM 网的关键活动为 a1, a5, a7, a8, a10, a12, a13 和 a14。

去掉网中的所有非关键路径，则变成下列有向图，如图 11.8-2 所示。

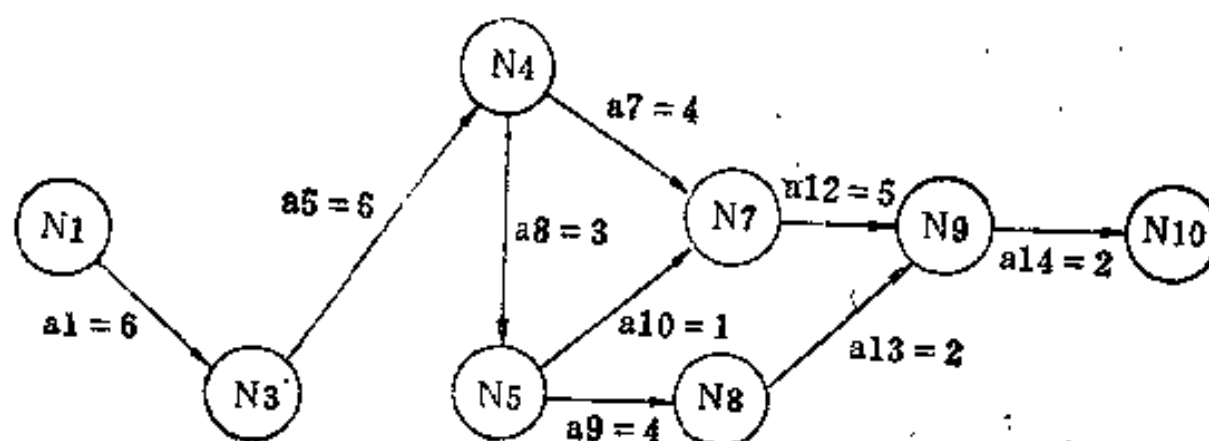


图 11.8-2 某工程的关键路径有向图

在图11.8-2中，由 N1至 N10的路径均为关键路径。此网中由 N4至 N9形成两条关键路径 (N4, N7和 N9) 和 (N4, N5, N8和 N9)。

只有在这两条关键路径上同时提高活动速度，才能缩短工期。在非关键路径上提高工效的努力，对缩短完成整个工程的时间是毫无效果的。

```

PROGRAM TOTAL8(INPUT, OUTPUT),
LABEL
  10;
CONST
  N=10; M=14;
TYPE
  ORLINK='ORTHNODE;
  ORTHNODE=RECORD
    TAIL, HEAD:INTEGER;
    DUT:INTEGER;
    HLINK, TLINK:ORLINK
  END;
  ORTHLIST=ARRAY (1..N) OF ORTHNODE;
VAR
  DIA:ORTHLIST;
  VE, VL:ARRAY (1..N) OF INTEGER;
  E, L:ARRAY (1..M) OF INTEGER;
  I, J, K, S, R, A:INTEGER;
  Q, P:ORLINK;
  C, B:TEXT;
BEGIN
  RESET(C, 'AOE.DAT');
  REWRITE(B, 'TOTAL8.DAT');
  10:
  FOR I:=1 TO N DO
    BEGIN

```

```

DIA (I) .HEAD:=0,
DIA (I) .TAIL:=0,
DIA (I) .DUT:=0,
DIA (I) .HLINK:=NIL,
DIA (I) .TLINK:=NIL
END,
FOR I:=1 TO M DO
BEGIN
  READ(C, J, K, A),
  DIA (J) .TAIL:=DIA (J) .TAIL+1,
  DIA (K) .HEAD:=DIA (K) .HEAD+1,
  NEW(Q),
  Q*.HEAD:=K, Q*.TAIL:=J,
  Q*.DUT:=A, Q*.HLINK:=DIA (K) .HLINK,
  Q*.TLINK:=DIA (J) .TLINK,
  DIA (K) .HLINK:=Q, DIA (J) .TLINK:=Q
END,
S:=1,
FOR I:=1 TO N DO
  VE (I) :=0,
  I:=0,
  WHILE S<>0 DO
    BEGIN
      J:=S, S:=DIA (J) .HEAD, Q:=DIA (J) .TLINK, I:=I+1,
      WHILE Q<>NIL DO
        BEGIN
          K:=Q*.HEAD, DIA (K) .HEAD:=DIA (K) .HEAD-1,
          IF DIA (K) .HEAD=0
            THEN
              BEGIN
                DIA (K) .HEAD:=S, S:=K
              END,
          IF VE (J) +Q*.DUT>VE (K)
            THEN VE (K) :=VE (J) +Q*.DUT,
          Q:=Q*.TLINK
        END,
      END,
    END,
  IF I<N
    THEN
      BEGIN
        WRITELN('THIS NETWORK HAS A CYCLE'),
        WRITELN(B,'THIS NETWORK HAS A CYCLE'), GOTO 16
      END,
  S:=N,

```

```

FOR I:=1 TO N DO
VL (I) :=VE (N) ;
WHILE S<>0 DO
BEGIN
K:=S; S:=DIA (K) .TAIL; Q:=DIA (K) .HLINK;
WHILE Q<>NIL DO
BEGIN
J:=Q*.TAIL; DIA (J) .TAIL:=DIA (J) .TAIL-1;
IF DIA (J) .TAIL=0
THEN
BEGIN
DIA (J) .TAIL:=S; S:=J
END;
IF VL (K) -Q*.DUT<VL (J)
THEN VL (J) :=VL (K) -Q*.DUT;
Q:=Q*.HLINK
END
END;
R:=0;
WRITELN(' ' 'INDEX TAIL HEAD ETIME LTIME DUT KEY');
WRITELN(B ' ' 'INDEX TAIL HEAD ETIME LTIME DUT KEY');
FOR I:=1 TO N DO
BEGIN
Q:=DIA (I) .TLINK;
WHILE Q<>NIL DO
BEGIN
J:=Q*.TAIL; K:=Q*.HEAD; R:=R+1;
E (R) :=VE (J) ; L (R) :=VL (K) -Q*.DUT;
IF E (R) =L (R)
THEN
BEGIN
WRITELN(R:7, J:5, K:5, E (R) :5, L (R) :5, Q*.DUT:
5,' *');
WRITELN(B, R:7, J:5, K:5, E (R) :5, L (R) :5, Q*.DUT:
5,' *');
END
ELSE
BEGIN
WRITELN(R:7, J:5, K:5, E (R) :5, L (R) :5, Q*.DUT:5);
WRITELN(B, R:7, J:5, K:5, E (R) :5, L (R) :5, Q*.DUT:5)
END;
Q:=Q*.TLINK
END;
END;
END;

```

```

WRITELN('NOTE:THE CRITICAL ACTIVITY HAS A • AT THE
      END. '),
WRITELN,
WRITELN(B,'NOTE:THE CRITICAL ACTIVITY HAS A • AT THE
      END. '),
WRITELN(B),
WRITELN(' ':10, 'ADJACENCY LIST'),
WRITELN(B,' ':10, 'ADJACENCY LIST'),
FOR I:=1 TO N DO
  BEGIN
    P:=DIA (I) .TLINK;
    WHILE P<>NIL DO
      BEGIN
        WRITE('(' ,P^.TAIL:2, P^.HEAD:3, P^.DUT:3,')...>'),
        WRITE(B,'(' ,P^.TAIL:2, P^.HEAD:3, P^.DUT:3,')...>'),
        P:=P^.TLINK
      END,
    IF I<N
      THEN
        WRITELN('LIN'),
        WRITELN(B,' LIN')
      END,
    WRITELN,
    WRITELN(B),
    WRITELN(' ':10, 'INVERSE ADJACENCY LIST'),
    WRITELN(B,' ':10, 'INVERSE ADJACENCY LIST'),
    FOR I:=1 TO N DO
      BEGIN
        Q:=DIA (I) .HLINK;
        WHILE Q<>NIL DO
          BEGIN
            WRITE('(' ,Q^.TAIL:2, Q^.HEAD:3, Q^.DUT:3,')...>'),
            WRITE(B,'(' ,Q^.TAIL:2, Q^.HEAD:3, Q^.DUT:3,')...>'),
            Q:=Q^.HLINK
          END,
        IF I>1
          THEN
            WRITELN(' NIL'),
            WRITELN(B,' NIL')
          END,
        CLOSE(B)
      END.
    (• - AOE.DAT •)
    { J K A }

```


1	2	5
1	3	6
2	4	3
3	4	6
3	5	3
4	5	3
4	7	4
5	7	1
5	8	4
4	6	4
6	10	4
7	9	5
8	6	2
9	10	2

OUTPUT:

(• TOTALs.DAT •)						
INDEX	TAIL	HEAD	ETIME	LTIME	DUT	KEY
1	1	3	0	0	6	•
2	1	2	0	4	5	
3	2	4	5	9	3	
4	3	6	6	12	3	
5	3	4	6	8	6	•
6	4	6	12	15	4	
7	4	7	12	12	4	•
8	4	5	12	12	3	•
9	5	8	15	15	4	•
10	5	7	15	15	1	•
11	6	10	16	19	4	
12	7	9	16	16	5	•
13	8	9	19	19	2	•
14	9	10	21	21	2	•

NOTE:THE CRITICAL ACTIVITY HAS A • AT THE END.

ADJACENCY LIST

```

(1 3 6) ———> (1 2 5) ———> LIN
(2 4 3) ———> LIN
(3 5 3) ———> (3 4 6) ———> LIN
(4 6 4) ———> (4 7 4) ———> (4 5 3) ———> LIN
(5 8 4) ———> (5 7 1) ———> LIN
(6 10 4) ———> LIN
(7 9 5) ———> LIN
(8 9 2) ———> LIN
(9 10 2) ———> LIN
LIN

```

INVERSE ADJACENCY LIST

```

      NIL
( 1 2 5 ) ----> NIL
( 1 3 6 ) ----> NIL
( 3 4 6 ) ----> ( 2 4 3 ) ----> NIL
( 4 5 3 ) ----> ( 3 5 3 ) ----> NIL
( 4 6 4 ) ----> NIL
( 5 7 1 ) ----> ( 4 7 4 ) ----> NIL
( 5 8 4 ) ----> NIL
( 8 9 2 ) ----> ( 7 9 5 ) ----> NIL
( 9 10 2 ) ----> ( 6 10 5 ) ----> NIL
FIG TOTAL8      PROGRAM

```

第九个程序 (TOTAL9.PAS)

功能：根据某大学1980年研究生入学考试的基本资料文件 GR.DAT，求出每一个专门化的考生的五门课程（政治、外语、基础课、技术基础课和专业课）的平均成绩，后三门主课的总分。根据五门平均成绩和后三门主课总分，按分数高低排队定出名次，并根据教育部的有关规定，决定录取与否。同时就该专业的全部考生的年龄，各门课程，五门平均分和三门主课总分等求其对应的平均数。最后将其统计表输出。

处理研究生入学考试基本资料文件，通常按专业统计，也可以按系统计，甚至可以全校一次统计。为了这个缘故，把统计处理放在过程 PROCEDURE TJ (N1,N2:INTEGER) 之中。

本程序的特点是采用了气泡沉浮法，就研究生入学考试五门平均分和三门主课总分按分数高低进行排队，排出名次，并按有关规定决定是否录取。气泡沉浮法，是管理统计程序中经常用到的一种排队方法。例如对人员基本资料的统计，或按年龄排队，或按成绩排队，或按其它具有某些特征的资料排队等等。

本程序中有两个程序设计技巧：一是处理大于计算机允许范围的整数所用的特殊方法；二是对某些特征数字要输出相应的文字字符的处理方法。

本程序研究生准考证号从801101至815405，显然超出 PDP-11/03 计算机系统允许的整数范围，在建立研究生基本资料文件 GR.DAT 时，准考证号是从1101至15405，而在处理后建立输出文件时相应加上'80'（在输入准考证号小于10000时）或'8'（在输入准考证号大于等于10000时）。

在建立基本资料文件时，为方便起见，对性别这一项，用整数1代表男性，0代表女性。而在输出统计表时，显然不能作这样的简化，对应代表男性的1用 MALE 表示，对应女性0的地方用 FEMALE 表示。在对每个考生的成绩统计后，按有关标准决定是否录取时，显然也不能再用数字来表示，应该用字符 YES 表示录取，NO 表示不录取更合适一些。

本程序中，必须提醒读者，对数据文件处理后输出的次序，已经是经过排队以后的顺序，而不是输入数据文件时的原始顺序。因此，在处理程序部分，必须注意在各项统计中凡是与序号 [I] 有关的地方，改为 X(I)，就是说有关统计项目都应该跟着按五门课程平均分和三门主课总分排队的次序走，而不能跟着原始输入文件的顺序去安排。否则会闹出许多笑话。

有关研究生的基本资料输入数据文件和处理以后的输出统计表以及源程序，见 FIG TOTAL9 PROGRAM。

```

PROGRAM TOTAL9(INPUT, OUTPUT);
(* 1980 GRADUATE STATISTICAL TABLE
FOR ENTRANCE EXAMINATION *)
CONST
    N=784;
TYPE
    AN=ARRAY [1..N] OF INTEGER;
    AH=PACKED ARRAY [1..40] OF CHAR;
    MN=ARRAY (1..3) OF CHAR;
VAR
    M1, M2:TEXT;
    CH:CHAR;
    GR:ARRAY (1..N, 1..10) OF INTEGER;
    X, Y, X1, Y1:AN;
    B3, B6, B7, B8, B9, B10, B11, B12, N1, N2, I, J, K:INTEGER;
    LQ:ARRAY (1..N) OF MN;
    CH1:AH;
    PROCEDURE RANGE(VAR X0, Y0:AN);
    BEGIN
        I:=N1+1;
        WHILE I<=N2 DO
            BEGIN
                J:=I;
                WHILE J>=N1+1 DO
                    BEGIN
                        IF Y0 (J) > Y0 (J-1)
                        THEN
                            BEGIN
                                K:=Y0 (J); Y0 (J) :=Y0 (J-1); Y0 (J-1) :=K;
                                K:=X0 (J); X0 (J) :=X0 (J-1); X0 (J-1) :=K;
                                J:=J-1;
                            END
                        ELSE EXIT
                        END;
                I:=I+1
            END;
        END;
    PROCEDURE TJ(N1, N2:INTEGER);
    BEGIN
        REWRITE(M2, 'LP:');
        FOR I:=N1 TO N2 DO
            FOR J:=1 TO 10 DO
                READ (M1, GR (I, J));
            FOR I:=N1 TO N2 DO

```

```

BEGIN
  X (I) := I, X1 (I) := I,
  K:=0,
  FOR J:=6 TO 10 DO K:=K+GR (I, J) ;
  Y (I) :=ROUND (K/5);
  Y1 (I) :=GR (I, 8) +GR (I, 9) +GR (I, 10) ;
  IF (Y (I) >=60) AND (Y1 (I) >=180)
    THEN LQ (I) :='YES'
    ELSE LQ (I) :='NO',
  END,
  RANGE (X, Y), RANGE (X1, Y1),
  B3:=0, B6:=0, B7:=0, B8:=0, B9:=0,
  B10:=0, B11:=0, B12:=0,
  FOR I:=N1 TO N2 DO
    BEGIN
      B3 :=B3 +GR (I, 3) ,      B6 :=B6 +GR (I, 6) ,
      B7 :=B7 +GR (I, 7) ,      B8 :=B8 +GR (I, 8) ,
      B9 :=B9 +GR (I, 9) ,      B10:=B10+GR (I, 10) ,
      B11:=B11+Y (I) ,          B12:=B12+Y1 (I)
    END,
    B3 :=B3 DIV (N2-N1+1),      B6 :=B6 DIV (N2-N1+1),
    B7 :=B7 DIV (N2-N1+1),      B8 :=B8 DIV (N2-N1+1),
    B9 :=B9 DIV (N2-N1+1),      B10 :=B10 DIV (N2-N1+1),
    B11:=B11 DIV (N2-N1+1),      B12 :=B12 DIV (N2-N1+1),
    WRITE (M2, ' ':10, ' • 1980 GRADUATE STATISTICAL TABLE'),
    WRITELN (M2, ' FOR ENTRANCE EXAMINATION • '),
    WRITELN (M2),
    WRITE(M2, 'DEPART—SPECIALITY:', CH1),
    WRITELN(M2, 'NUMBER:', N1:3, '...', N2:3),
    WRITELN(M2),
    WRITE (M2, ' 1 2 3 4 5 6 7 8'),
    WRITELN (M2, ' 9 10 11 12 13 14'),
    WRITE (M2, ' NUM PAPERS AGE SEX GYEAR POLI FORE BASE'),
    WRITELN(M2, ' TECH SPEC (6-10)/5 8+9+10 ORD ENROLL? '),
    WRITELN (M2),
    FOR I:=N1 TO N2 DO
      BEGIN
        WRITE (M2, X (I) :3);
        FOR J:=2 TO 10 DO
          BEGIN
            CASE J OF
              2:
                BEGIN
                  IF GR (X (I) , J) <10000

```

```

        THEN WRITE(M2, ' 80', GR (X (I) , J) :4);
        IF GR (X (I) , J) >=10000
        THEN WRITE(M2, ' 8', GR (X (I) , J) :5)
    END,
3: WRITE (M2, GR (X (I) , J) :4);
4:
    BEGIN
        IF GR (X (I) , J) =1
        THEN WRITE (M2, ' MALE');
        IF GR (X (I) , J) =0
        THEN WRITE (M2, ' FEMA')
    END,
5: WRITE (M2, GR (X (I) , J) :7);
    ELSE WRITE (M2, GR (X (I) , J) :6)
    END
END,
    WRITELN (M2, Y (I) :8, Y1 (I) :9, (I-N1+1):7, LQ (X (I) ) :9)
END,
    WRITELN (M2);
    WRITE (M2, 'AVERAGE' , B3:8, B6:20, B7:6, B8:6, B9:6);
    WRITELN (M2, B10:6, B11:8, B12:9); WRITELN (M2); WRITELN (M2);
    CLOSE (M2)
END,
BEGIN,
    CH:='N',
    RESET (M1, 'GR.DAT');
    REPEAT
        WRITELN ('PLEASE INPUT DEPART-SPECIALITY: ');
        READLN (CH1);
        WRITELN ('PLEASE INPUT N1=? '); READLN (N1);
        WRITELN ('PLEASE INPUT N2=? '); READLN (N2);
        TJ (N1, N2);
        WRITELN ('FINISHED? (YES INPUT "Y" , ELSE INPUT "N" ) ');
        READLN(CH)
    UNTIL CH='Y'
END.

```

(• GR.DAT •)

001	01101	27	1	1978	33	31	13	18	46
002	01102	26	0	1980	50	70	66	71	75
003	01103	23	1	1980	00	38	25	26	00	770 15401 32 1 1975 52 53 26 31 41
004	01104	25	1	1979	44	37	24	37	78	771 15402 24 1 1980 52 83 38 10 52
005	01105	25	1	1979	60	34	26	33	69	772 15403 33 1 1970 66 66 63 60 40
006	01106	30	1	1976	31	22	37	43	56	773 15404 32 0 1970 67 64 45 30 37
007	01107	34	1	1970	50	72	65	99	61	774 15405 32 1 0000 46 66 62 65 85

OUTPUT: (•LP: •)
 • 1980 GRADUATE STATISTICAL TABLE FOR ENTRANCE EXAMINATION •
 DEPART-SPECIALITY:DEPT.OF.ARCHIT...ARCHITECTURAL.DESIGN NUMBER:1..7

1	2	3	4	5	6	7	8	9	10	11	12	13	14
NUM	PAPERS	AGE	SEX	GYEAR	POLI	FORE	BASE	TECH	SPEC	(6-10)/5	8+9+10	ORD	ENROLL†
7	801107	34	MALE	1970	50	72	65	99	61	69	225	1	YES
2	801102	26	FEMA	1980	50	70	66	71	75	66	212	2	YES
4	801104	25	MALE	1979	44	37	24	37	78	44	139	3	NO
5	801105	25	MALE	1979	60	34	26	33	69	44	136	4	NO
6	801106	30	MALE	1976	31	22	37	43	36	38	128	5	NO
1	801101	27	MALE	1978	33	31	13	18	46	28	77	6	NO
3	801103	23	MALE	1980	0	38	25	26	0	18	51	7	NO
AVERAGE		27			38	43	36	46	55	43	138		

.....

• 1980 GRADUATE STATISTICAL TABLE FOR ENTRANCE EXAMINATION •
 DEPART-SPECIALITY: DEPT.OF.AUTOMATION-COMPUTER NUMBER:770..774

1	2	3	4	5	6	7	8	9	10	11	12	13	14
NUM	PAPERS	AGE	SEX	GYEAR	POLI	FORE	BASE	TECH	SPEC	(6-10)/5	8+9+10	ORD	ENROLL
774	815405	32	MALE	0	46	66	62	65	85	65	212	1	YES
772	815403	33	MALE	1970	66	66	63	60	60	63	183	2	YES
773	815404	32	FEMA	1970	67	64	45	30	37	49	112	3	NO
771	815402	24	MALE	1980	52	83	38	10	52	47	100	4	NO
770	815401	32	MALE	1975	52	53	26	31	41	41	98	5	NO
AVERAGE		30			56	66	46	39	55	53	141		

FIG TOTAL9 PROGRAM

第十个程序 (TOTL10.PAS)

功能：对1980年研究生入学考试基本资料文件 GR.DAT 进行专项统计：计算并输出某专业、某系或全校的参加入学考试的研究生应试者按年龄、按毕业年份的分配人数及所占的百分比。

GR.DAT 还是第九个程序中的那个输入数据文件。在情况统计和管理工作中，某些基本情况构成的数据文件，是共享的资源，对其可以进行不同的处理。例如一个学校人员的基本情况，可以构成一个单独的数据文件，存放在盘中。人们可以编制不同的程序，按需要进行不同的统计、汇总、分类、排队等等，并将其处理结果建立一个或若干个新的数据文件，显示或打印出来。本程序就是为解决这类问题向读者介绍的一个简单应用程序。

本程序结构清晰，简明扼要。有以下特点：

第一，本程序可以对任意区段的资料进行统计，既可以重复，也可以交叉，例如在对序号从1至100的研究生应试者进行统计后，可以接下去从序号200至500进行统计，也可以重新由序号1至300，或1至784全部人数进行统计。为什么能这样呢？原因是把读基本资料文件 GR.DAT 的数据，放在主程序部分，一次全部读入，然后按要求的区段的数字在过程中进行统计（在第五个程序中，也采取了这个办法）；

第二，在统计和输出的处理上都用了循环语句按顺序进行的办法实现，所以使程序特别简单；

第三，设置特殊数组，以减少程序运行时对内存的要求。本程序中，在进行毕业年份统计中，遇到同等学历的这一挡（用0表示），如果要把0到1980作为下标设置数组，那末它为包含1981项由整数构成的数组，这样必然要占据太多的内存单元，而且绝大多数是无用的。本程序中只设置了从1963到1980作为下标的数组，另外单独设置了(0..0)作为下标的特殊数组，用来表示同等学历的毕业年份。这样用于按毕业年份统计人数的数组仅包括19项，不及1981项的百分之一。这不仅可以节省内存单元，也可使统计的时间大为缩短。这种设置数组的情况是会经常遇到的；

第四，保证使输出格式整齐，本程序中为了使有关人数的百分比符号（%）以及小数点对齐，对输出场宽进行了调整。如果在10%以下，输出有效数字从0.0到9.9只有三列，而10%以上，输出数字从10.0到99.9为四列。在这种情况下不能用同一场宽输出，而必须进行相应调整，以保证输出格式的整齐。

有关的源程序，输入数据文件和输出统计文件见 FIG TOTAL10 PROGRAM.

```
PROGRAM TOTAL10 (INPUT, OUTPUT);
CONST
  N=784;
TYPE
  AH=ARRAY [1..N] OF INTEGER;
  AH=PACKED ARRAY [1..25] OF CHAR;
VAR
  I, J, K, M, N1, N2:INTEGER;
```

```

CH:CHAR;
CH1:AH;
GR:ARRAY (1..N, 1..10) OF INTEGER;
NUMBER:ARRAY (19..38) OF INTEGER;
SUM: ARRAY (1963..1980) OF INTEGER;
SUM1:ARRAY (0..0) OF INTEGER;
P, Q:TEXT;
PROCEDURE TJ (N1, N2:INTEGER);
  BEGIN
    REWRITE (P, 'LP:');
    FOR K:=19 TO 38 DO NUMBER (K) :=0;
    FOR M:=1963 TO 1978 DO SUM (M) :=0;
    SUM1 (0) :=0;
    FOR I:=N1 TO N2 DO
      BEGIN
        K:=GR (I, 3) ;
        NUMBER (K) :=NUMBER (K) +1;
        M:=GR (I, 5) ;
        IF M=0
          THEN SUM1 (0) :=SUM1 (0) +1
          ELSE SUM (M) :=SUM (M) +1
        END;
      WRITE (P, ' • 1980 GRADUATE STATISTICAL TABLE');
      WRITELN (P);
      WRITELN (P, 'FOR AGE AND GRADUATE YEAR •');
      WRITELN (P);
      WRITELN (P, 'UNIT NAME: ', CH1, N1:3, '...', N2:3);
      WRITELN (P);
      WRITELN (P, ' ':12, ' • AGE STATISTICAL TABLE •');
      WRITELN (P);
      WRITELN (P, ' ':15, 'AGE      NUM      PERC');
      WRITELN (P);
      FOR K:=19 TO 38 DO
        BEGIN
          WRITE (P, ' ':15, K:2, '...', NUMBER (K) :3, ' ':5);
          IF NUMBER (K) /(N2-N1+1)<0.10
            THEN WRITELN(P, ' ', NUMBER (K) *100/(N2-N1+1):3:1, '%');
          IF NUMBER (K) / (N2-N1+1)>=0.10
            THEN WRITELN(P, NUMBER (K) *100/(N2-N1+1):4:1, '%')
          END;
        WRITELN (P);
        WRITELN (P);
        WRITELN (P, ' ':12, ' • GRADUATE YEAR STATISTICAL TABLE •');
        WRITELN (P);
      END;
    END;
  END;

```



```

WRITELN (P, ' ':15, 'YEAR          NUM          PERC');
WRITELN (P);
FOR M:=1963 TO 1980 DO
  BEGIN
    WRITE (P, ' ':15, M:4'...', SUM (M) :3, ' ':5);
    IF SUM (M) /(N2-N1+1)<0.10
      THEN WRITELN(P, ' ', SUM (M) * 100/(N2-N1+1):3:1, '%');
    IF SUM (M) /(N2-N1+1)>=0.10
      THEN WRITELN(P, SUM (M) * 100/(N2-N1+1):4:1, '%')
    END;
  WRITE (P, ' ':15, '0000', '...', SUM1 (0) :3);
  WRITELN (P, ' ':6, SUM1 (0) * 100/(N2-N1+1):4:1, '%');
  WRITELN (P);
  WRITELN (P);
  CLOSE (P)
  END;
BEGIN
  CH:='N',
  RESET (Q, 'GR');
  FOR I:=1 TO 784 DO
    FOR J:=1 TO 10 DO
      READ (Q, GR (I, J) );
      REPEAT
        WRITELN ('PLEASE INPUT UNIT NAME ', CH); READLN (CH);
        WRITELN ('PLEASE INPUT NUMBER N1=? '); READLN(N1);
        WRITELN ('PLEASE INPUT NUMBER N2=? '); READLN(N2);
        TJ (N1, N2);
        WRITELN ('FINISHED? (YES, INPUT "Y", ELSE INPUT "N")');
        READLN (CH);
      UNTIL CH='Y'
    END.
  ( •      GR.BAT      • )
001 01101 27 1 1978 33 31 13 18 46
002 01102 26 0 1980 50 70 66 71 75
.....
783 15507 32 1 1970 67 61 57 87 69
784 15508 18 1 1980 26 47 87 45 64
OUTPUT: ( •  LP: • )
• 1980 GRADUATE STATISTICAL TABLE FOR AGE AND GRADUATE
  YEAR •
UNIT NAME:DEPT.OF.ARCHITECTURE 1..100
  • AGE STATISTICAL TABLE •
    AGE      NUM      PERC
    19      ...      0      0.0%

```

20	...	0	0.0%
21	...	0	0.0%
22	...	0	0.0%
23	...	4	4.0%
24	...	5	5.0%
25	...	12	12.0%
26	...	12	12.0%
27	...	8	8.0%
28	...	6	6.0%
29	...	5	5.0%
30	...	8	8.0%
31	...	7	7.0%
32	...	4	4.0%
33	...	4	4.0%
34	...	9	9.0%
35	...	6	6.0%
36	...	5	5.0%
37	...	4	4.0%
38	...	1	1.0%

• GRADUATE YEAR STATISTICAL TABLE •

YEAR		NUM	PERC
1963	...	0	0.0%
1964	...	2	2.0%
1965	...	2	2.0%
1966	...	1	1.0%
1967	...	0	0.0%
1968	...	10	10.0%
1969	...	1	1.0%
1970	...	12	12.0%
1971	...	0	0.0%
1972	...	0	0.0%
1973	...	3	3.0%
1974	...	0	0.0%
1975	...	8	8.0%
1976	...	7	7.0%
1977	...	12	12.0%
1978	...	13	13.0%
1979	...	16	16.0%
1980	...	10	10.0%
0000	...	3	3.0%

• 1980 GRADUATE STATISTICAL TABLE FOR AGE AND GRADUATE YEAR •

UNIT NAME:TOTAL.OF.UNTVERSITY 1..784

• AGE STATISTICAL TABLE •

AGE		NUM	PERC
19	...	1	0.1%
20	...	1	0.1%
21	...	3	0.4%
22	...	5	0.6%
23	...	12	1.5%
24	...	40	5.1%
25	...	83	10.6%
26	...	93	11.9%
27	...	74	9.4%
28	...	51	6.5%
29	...	36	4.6%
30	...	45	5.7%
31	...	36	4.6%
32	...	35	4.5%
33	...	48	6.1%
34	...	96	12.2%
35	...	68	8.7%
36	...	21	2.7%
37	...	29	3.7%
38	...	7	0.9%

• GRADUATE YEAR STATISTICAL TABLE •

YEAR		NUM	PERC
1963	...	2	0.3%
1964	...	3	0.4%
1965	...	5	0.6%
1966	...	4	0.5%
1967	...	5	0.6%
1968	...	93	11.9%
1969	...	4	0.5%
1970	...	136	17.3%
1971	...	0	0.0%
1972	...	0	0.0%
1973	...	4	0.5%
1974	...	9	1.1%
1975	...	48	6.1%
1976	...	58	7.4%
1977	...	91	11.6%
1978	...	88	11.2%
1979	...	82	10.5%
1980	...	125	15.9%
0000	...	27	3.4%
FIG	TOTAL	10	PROGRAM

第十一个程序 (TOTAL11.PAS)

功能：用列表法单输出布尔函数的化简。

布尔函数的人工化简方法有公式法和图解法。这在变量较少时是常常采用的。但在变量较多的情况下，用计算机来化简布尔函数是最有效的方法。

多输出函数的化简，要比单输出函数的化简具有更大的实用价值。但作为一个例子，为更清楚地了解布尔函数的计算机化简方法，还是选择单输出布尔函数的化简程序为好。

这个程序由主程序和两个过程组成。主程序的开始部分是确定布尔函数的输入输出数据文件的名称，然后进行以下工作：

1. 从磁盘上存放的输入文件向内存调入原始数据；
2. 对输入的最小项、不顾项按递增顺序排队；
3. 求函数的假值（即产生OFF-ARRAY），存放在最小项、不顾项数组后半部；
4. 求函数的全部质蕴涵项；
5. 挑选必要的质蕴涵项或极值项；
6. 挑选一个非极值项。

直到选出的质蕴涵项能复盖函数的全部最小项，程序结束。

程序中含有二个过程。一个是PROCEDURE DELPI，它的功能是从质蕴涵项表中，删除劣势的质蕴涵项，修改该质蕴涵项所包含的最小项的复盖次数。另一个过程是PROCEDURE C14，它是在选中极值项或选中一个非极值蕴涵项后调用的过程，它的主要功能是在最小项数组里删除被选中的质蕴涵项所包含的最小项，在质蕴涵项数组里删除被选中的质蕴涵项，然后以立方体形式输出被选中的质蕴涵项。

本程序列举了处理两个不同的输入数据文件TO11.DAT和TOT11.DAT以及输出相应的化简结果输出数据文件OTO11.DAT和OTOT11.DAT。有关的源程序，输入输出数据文件见FIG TOTAL11 PROGRAM。

```
PROGRAM TOTAL11(INPUT, OUTPUT);
  { SINGLE OUTPUT BOOLEAN FUNCTION IS MINIMIZED }
LABEL
  5, 10 100;
CONST
  C=4096;    B=4;
TYPE
  ONE=ARRAY [1..C] OF INTEGER;
  TWO=ARRAY [1..500, 1..4] OF INTEGER;
VAR
  FI, FO: TEXT;
  CHARS, NAME: ALFA;
  G, F, W, NC, A9, B9, Q MI, V VN MN, NCN, X, Y, K, I, J, Z, L1, L2:
    INTEGER;
  N1, N, H:1..C; D, M:ONE; D1:ARRAY [1..100] OF INTEGER;
  P:TWO;
  CH:CHAR;
  BO:BOOLEAN;
```

```

PROCEDURE DELP1;
BEGIN
  FOR I:=N1 DOWNTO 1 DO
    BEGIN
      Q:=0; N:=0;
      FOR K:=1 TO X DO
        IF (P (I, 1) AND M (K) =P (I, 1) ) AND
          (P (I, 2) AND M (K) =M (K) )
          THEN
            BEGIN
              Q:=Q+1; N:=N+1; D1 (Q) :=K
            END;
      IF N=0
      THEN
        BEGIN
          P (I, 1) :=P (N1, 1) ; P (I, 2) :=P (N1, 2) ; P (I, 3) :=P (N1, 3) ;
          P (I, 4) :=P (N1, 4) ; N1:=N1-1;
        END
      ELSE
        FOR J:=1 TO N1 DO
          IF I<>J
          THEN
            BEGIN
              FOR K:=1 TO N DO
                IF (P (J, 1) AND M (D1 (K) ) =P (J, 1) ) AND
                  (P (J, 2) AND M (D1 (K) ) =M (D1 (K) ) )
                  THEN
                    ELSE EXIT;
                IF (K=N+1) AND (P (I, 3) <=P (J, 3) )
                THEN
                  BEGIN
                    IF I<>N1
                    THEN
                      BEGIN
                        P (I, 1) :=P (N1, 1) ; P (I, 2) :=P (N1, 2) ;
                        P (I, 3) :=P (N1, 3) ; P (I, 4) :=P (N1, 4)
                      END;
                        N1:=N1-1;
                        FOR Q:=1 TO N DO D (D1 (Q) ) :=D (D1 (Q) ) -1;
                      EXIT
                    END
                  END
                END;
            END;
          GOTO 100

```

```

END,
PROCEDURE C14(VAR H, L1, I, J, Z, L2:INTEGER),
BEGIN
  Q:=1;
  REPEAT
    IF (I AND M(Q) = I) AND (J AND M(Q) = M(Q))
    THEN
      BEGIN
        M(Q) := M(H), D(Q) := D(H), H:=H-1, Q:=Q-1
      END,
      Q:=Q+1
    UNTIL Q=H+1,
  K:=B9,
  WRITE(I:6, J),
  WRITE(FO, I:6, J),
  WRITE(' '),
  WRITE(FO, ' '),
  FOR Q:=1 TO V DO
    BEGIN
      K:=TRUNC(K/2),
      IF I-K >= 0
      THEN
        BEGIN
          I:=I-K, J:=J-K, CH:='1', WRITE(CH),
          WRITE(FO, CH)
        END
      ELSE
        IF J-K < 0
        THEN
          BEGIN
            CH:='0', WRITE(CH), WRITE(FO, CH)
          END
        ELSE
          BEGIN
            CH:='-', J:=J-K, WRITE(CH), WRITE(FO, CH)
          END
        END
      END,
  WRITELN,
  WRITELN(FO),
  IF N=L1
  THEN L1:=L1-1
  ELSE
  BEGIN
    I:=P(L1, 1), Z:=P(L1, 3),

```

```

      J:=P (L1, 2) ; L2:=P (L1,4) ,
      L1:=L1-1
    END
  END;
BEGIN (• MAIN PROGRAM •)
  WRITE ('OUTPUT FILE NAME ? ');
  READ (NAME);
  REWRITE (FO, NAME);
  WRITE ('INPUT FILE NAME ? ');
  READ (NAME);
  RESET (FI, NAME);
  WHILE NOT EOF (FI) DO
    BEGIN
      READLN (FI, CHARS); Z:=0;
      FOR I:=1 TO 10 DO Z:=Z+ORD(CHARS (I) );
      CASE Z OF
        748 :READ (FI, V);
        743 :READ (FI, X);
        662 :FOR H:=1 TO X DO
          BEGIN
            M (H) :=0; D (H) :=0; READ (FI, M (H) ); D (H) :=M (H)
          END;
        682 :READ (FI, Y);
        689 :FOR H:=1 TO Y DO
          BEGIN
            D (X+H) :=0; READ (FI, D (X+H) )
          END;
      END
    END
  END;
  A9:=X+Y;
  Q:=V;
  B9:=1
  WHILE Q<>0 DO
    BEGIN;
      B9:=B9*2; Q:=Q-1
    END;
  10:
  FOR J:=1 TO (A9-1) DO
    BEGIN
      K:=J;
      REPEAT
        IF D (K) >D (K+1)
          THEN
            BEGIN

```

```

      Z:=D (K) ;
      D (K) :=D (K+1) ;
      D (K+1) :=Z
    END,
    K:=K-1
  UNTIL K=0
END,
K:=0,
REPEAT
  FOR H:=1 TO A9 DO
    IF K=D (H)
      THEN GOTO 5,
    A9:=A9+1,
    D (A9) :=K,
5:
    K:=K+1
  UNTIL K=B9,
  A9:=X+Y,
  FOR N:=1 TO 500 DO
    BEGIN
      P (N, 1) :=0, P (N, 2) :=0, P (N, 3) :=0, P (N, 4) :=0
    END,
    N:=1,
    FOR L1:=1 TO A9 DO
      BEGIN
        I:=D (L1) ,
        FOR L2:=A9 DOWNT0 L1 DO
          BEGIN
            J:=D (L2) ,
            IF I AND J=I
              THEN
                BEGIN
                  FOR K:=(A9+1) TO B9 DO
                    IF (I AND D (K) =I) AND (D (K) AND J=D (K) )
                      THEN
                        EXIT,
                    IF K= (B9+1)
                      THEN
                        BEGIN
                          P (N, 1) :=I,
                          P (N, 2) :=J,
                          IF N=1
                            THEN
                              BEGIN

```



```

        N:=2, Z:=N
    END
ELSE
    BEGIN
        H:=N,
        REPEAT
            IF (P (H-1, 1) AND P (N, 1) <> P (H-1, 1) ) OR
                (P (H-1, 2) AND P (N, 2) <> P (N, 2) )
            THEN H:=H-1
            ELSE
                BEGIN
                    N:=N-1, EXIT
                END
            UNTIL H=1,
            N:=N+1, Z:=N
        END
    END
END
END
END,
N1:=N-1,
FOR H:=1 TO X DO
    BEGIN
        D (H) :=0, FOR N:=1 TO N1 DO
            IF (P (N, 1) AND M (H) =P (N,1) ) AND
                (P (N, 2) AND M (H) =M (H) )
            THEN D (H) :=D (H) +1
        END,
        WRITELN ('      MINIMIZED SOLUTION'),
        WRITELN (FO, '      MINIMIZED SOLUTION'),
        WRITELN ('      I      J      PRODUCT'),
        WRITELN (FO, '      I      J      PRODUCT'),
        DELPI,
100:
        BO:=FALSE,
        REPEAT
            Z:=X,
            FOR H:=1 TO Z DO
                IF D (H) =1
                THEN
                    BEGIN
                        BO:=TRUE,
                        FOR N:=1 TO N1 DO
                            IF (P (N, 1) AND M (H) =P (N,1) ) AND (P (N,2) AND

```

```

        M (H) = M (H) )
    THEN
        BEGIN
            C14 (X, N1, P (N, 1) , P (N, 2) , P (N, 3) , P (N, 4) ),
            EXIT
        END,
    EXIT
END
UNTIL H>Z,
IF (X<>0) AND (BO=TRUE)
    THEN DELP1,
IF X<>0
    THEN
        BEGIN
            F:=2,
            FOR N:=1 TO N1 DO P (N, 4) :=0,
            Q:=X,
            FOR H:=1 TO G DO
                IF D (H) =F
                    THEN
                        BEGIN
                            K:=0,
                            FOR N:=1 TO N1 DO
                                BEGIN
                                    IF (P (N, 1) AND (M (H) =P (N, 1) )
                                        AND (P (N, 2) AND (M (H) =M (H) )
                                    THEN
                                        BEGIN
                                            FOR J:=1 TO G DO
                                                IF (P (N, 1) AND M (J) =P (N, 1) )AND
                                                    (P (N, 2) AND M (J) =M (J) )
                                                    THEN P (N, 4) :=P (N, 4) +1,
                                                    K:=K+1, D1 (K) :=N
                                        END,
                                    IF K=F
                                        THEN EXIT
                                END,
                            FOR Q:=1 TO F-1 DO
                                BEGIN
                                    I:=Q,
                                    REPEAT
                                        IF P (D1 (Q) , 4) >P (D1 (Q+1) , 4)
                                            THEN
                                                BEGIN

```

```

      Z := D1 (Q) ; D1 (Q) := D1 (Q+1) ; D1 (Q+1) := Z
    END;
    I := I - 1
  UNTIL I = 0
END;
IF P (D1 (F) , 4) > P (D1 (F-1) , 4)
THEN
  BEGIN
    C14 (X, N1, P (D1 (F) , 1) , P (D1 (F) , 2) , P (D1 (F) , 3) ,
      P (D1 (F) , 4) );
    EXIT
  END
ELSE
  IF P (D1 (F-1) , 3) > P (D1 (F) , 3)
  THEN
    BEGIN
      C14 (X, N1, P (D1 (F-1) , 1) , P (D1 (F-1) , 2) , P (D1 (F-1) , 3) ,
        P (D1 (F-1) , 4) );
      EXIT
    END
  ELSE
    BEGIN
      C14 (X, N1, P (D1 (F) , 1) , P (D1 (F) , 2) , P (D1 (F) , 3) ,
        P (D1 (F) , 4) );
      EXIT
    END
  END;
  IF H > G
  THEN F := F + 1;
  IF X <> 0
  THEN DELPI
  END;
  CLOSE (FO)
END.
INPUT:
  (• TO11.DAT •)
VARNUMBER:
  4
MINNUMBER:
  8
MINTERM:
  0 1 3 4 5 13 15 17
DCNUMBER:
  0

```

DON'TCARE:

OUTPUT:

```
(•      OTO11.DAT      •)
MINIMIZED SOLUTION
I          J      PRODUCT
0          6      0-0-
3          11     -011
13         15     11-1
```

INPUT:

```
(•      TOT11.DAT      •)
```

VARNUMBER:

8

MINNUMBER,

43

MINTERM:

17 20 21 23 25 32 34 35 38 39 48 49 53 54 64 65 66 70 71
72 73 84 85 86 87 98 99 100 101 102 114 115 116 117 118 119
132 133 134 135 136 137 151

DCNUMBER:

153

DON'TCARE:

0 10 11 12 13 14 15 26 27 28 29 30 31 42 43 44 45 46
47 58 59 60 61 62 63 74 75 76 77 78 79 90 91 92 93 94
95 106 107 108 109 110 111 122 123 124 125 126 127 138
139 140 141 142 143 154 155 156 157 158 159 160 161 162
163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190
191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218
219 220 221 222 223 224 225 226 227 228 229 230 231 232
233 234 235 236 237 238 239 240 241 242 243 244 245 246
247 248 249 250 251

OUTPUT:

```
(•      OTOT11.DAT      •)
MINIMIZED SOLUTION
I          J      PRODUCT
20         93     0-01-10-
136        239     1--01---
132        239     1--0-1--
17         29     0001--01
34         175     -010--1-
100        245     -11-010-
38         246     --1-0110
64         201     -100-00-
```

70	223	- 1 0 - - 1 1 -
98	251	- 1 1 - - 0 1 -
23	223	- - 0 1 - 1 1 1
66	238	- 1 - 0 - - 1 0
32	176	- 0 1 - 0 0 0 0
17	53	0 0 - 1 0 - 0 1
114	247	- 1 1 1 0 - 1 0

FIG TOTAL11 PROGRAM

第十二个程序 (TOTL12.PAS)

功能：实现人与计算机下棋。

有关本例的源程序，棋盘数据文件，下棋过程等见FIG TOTA12 PROGRAM。

CHESS.CAT是下棋的棋盘，在下棋的过程中，它始终显示在终端屏幕上。

下棋由A（代表人）和B（代表计算机）两方进行，每方是两个子，棋盘上有五个位置，其编号分别为1，2，3，4和5。下棋开始前，A占1和2两个位置，B占4和5两个位置，3为空闲位置（FREE）。走子的规则，每方一次走一个子，双方轮换进行。走子时只能把自己的一个子沿直线方向走到相邻的空闲位置，不得在两个无连线的位置间走子（例如位置1和2），也不得跳过一个位置到下一个位置，如位置1和4，位置2和5。当轮到一方走子时，无路可走，就被认为是输了。这时，计算机在终端屏幕上显示一行信息：

YOU ARE FAILED, FOR A MOMENT GO ON,

具体走法是A先走。当计算机输出信息：

INPUT YOUR ANSWER,

之后，A把自己的走法通过终端告诉计算机。如A要从位置1走到位置3，就在键盘上打入：

1 3

即数字1键，空格键，数字3键和回车键。之后，计算机就自动走自己的一步，并在终端屏幕上重新显示双方走子后的棋盘上的新形势，如，

3位置被A占据；

1位置被B占据；

5位置变成空闲。

这就是人机下棋的第一步，接着可走第二步，第三步……，一直到A即人输掉为止。这个程序保证计算机不会输。

为了便于读者看懂这个程序，把程序的结构和各部分的功能简介如下。

一、在类型说明中，定义：

TYPE RANGE=1..5;

这是用来表达棋盘上的五个位置，其编号分别为1，2，3，4和5。

二、在变量说明部分，要解释以下几点：

1. TEXT文件F，对应实际的数据文件为CHESS.DAT，其内容就是人机下棋的棋盘；

2. 数组AR，是为了记忆棋盘上的形势用的。开始时，是把棋盘的初始形势读入数组，以后每走一步，就用它记住当时棋盘的形势。对用CHESS.DAT表示的初始棋盘形势始终不进行修改而原封不动保留着。在走棋过程中，只用数组AR，而不是TEXT文件F；

3. 数组A和B, 分别用来记忆A方和B方的两个子所在位置的编号。变量FREE用来记忆空闲位置的编号:

4. 数组MAP和PATH很重要。MAP用来给出棋盘上5个位置在数组AR中的相应的下标值, 以供每走完一步之后修正棋盘上的实际形势时使用。数组PATH用来记忆五个位置之间的连线关系。这里用的是二维数组, 如位置1和2之间无连线, 则让PATH(1, 2)和PATH(2, 1)为FALSE; 位置1和5之间有连线, 则让PATH(1, 5)和PATH(5, 1)为TRUE。PATH记忆各位置之间的通路关系, 计算机在走棋时要查询它。

三、本程序用了三个过程:

1. 过程PEAPLN, 检查人走子的合法情况, 如能走, 完成之; 如不能走, 则应指出其错误, 请重新走子;

2. 过程COMPUTER, 解决计算机如何判断走子。它首先判断哪个子能走而且不会输, 然后决定走这个子;

3. 过程MODIFY, 根据走子后的结果修改数组AR的内容, 也就是棋盘上的形势。

四、本程序的主程序分成四部分:

1. 第一部分, 对数组和变量赋初值。请注意弄清数组PATH和MAP赋予初值的意义;

2. 第二部分, 把棋盘文件的内容读到数组AR中来。请弄清它是如何协调文件F和数组AR的内容的;

3. 第三部分, 把数组AR中的内容输出到终端屏幕上显示。第二个IF语句, 检查下棋过程中人是否输了, 若是则输出一行信息: 你输了, 过一会儿再继续下棋。

为开始下一盘棋赛, 要重新从软磁盘中的数据文件CHESS.DAT里读数据到数组AR中去;

4. 第四部分, 先提示人要输入信息, 然后检查输入的内容是否正确, 若输入正确, 则调用过程PEAPLE; 否则要重新输入。人走完之后, 调用过程COMPUTER, 实现计算机走子。计算机走子后, 调用过程MODIFY, 修改并显示此时棋盘上的新形势。

```
PROGRAM TOTAL12 (INPUT, OUTPUT, F);
LABEL
  1, 2, 3;
TYPE
  RANGE=1..5;
VAR
  F:TEXT;
  I, J, S, D:INTEGER;
  AR:ARRAY [1..40, 1..70] OF CHAR;
  FREE:RANGE;
  A, B:ARRAY [1..2] OF RANGE;
  ST:SET OF RANGE;
  MAP:ARRAY [RANGE, 1..2] OF INTEGER;
  PATH:ARRAY [RANGE, RANGE] OF BOOLEAN;
  T:REAL;
  FIRST:BOOLEAN;
  PROCEDURE PEAPLE;
```

```

BEGIN
  IF ((S=A (1) ) OR (S=A (2) )) AND (D=FREE)
  THEN
    IF S=A (1)
    THEN A (1) :=D
    ELSE A (2) :=D
  ELSE
    BEGIN
      WRITELN ('YOU ARE NOT AT PLACE',S:2,'OR YOU CAN');
      WRITELN ('NOT GO TO PLACE', D:2, ', INPUT AGAIN ');
      GOTO 1
    END
  END,
PROCEDURE COMPUTER,
BEGIN
  FREE:=B (1) ,
  IF PATH (B (1) , S) AND (B (2) +S<>8)
  THEN B (1) :=S
  ELSE
    IF PATH (B (2) , S) AND (B (1) +S<>8)
    THEN
      BEGIN
        FREE:=B (2) , R (2) :=S
      END,
    END,
  END,
PROCEDURE MODIFY,
BEGIN
  FOR I:=1 TO S DO
    IF (A (1) =I) OR (A (2) =I)
    THEN AR (MAP (I, 1) , MAP (I, 2) ) :='A'
    ELSE
      IF (B (1) =I) OR (B (2) =I)
      THEN AR (MAP (I, 1) , MAP (I, 2) ) :='B'
      ELSE
        AR (MAP (I, 1) , MAP (I, 2) ) :=' ',
    END,
  END,
BEGIN (• MAIN PROGRAM •)
  (• PART 1...INITIALIZE SOME ARRAYS AND VARIABLES •)
  ST:= {1..5} ;
  FOR I:=1 TO 2 DO
    BEGIN
      A (I) :=1, B (I) :=I+3
    END,
  FREE:=3,

```

```

FOR I:=1 TO 5 DO
FOR J:=1 TO 5 DO
IF I=J
THEN PATH (I, J) :=FALSE
ELSE
IF (I=1) AND (J=2) OR (I=2) AND (J=1)
THEN PATH (I, J) :=FALSE
ELSE
IF (I=1) AND (J=4) OR (I=4) AND (J=1)
THEN PATH (I, J) :=FALSE
ELSE
IF (I=5) AND (J=2) OR (I=2) AND (J=5)
THEN PATH (I, J) :=FALSE
ELSE
PATH (I, J) :=TRUE,
MAP (1, 1) :=2 ,      MAP (1, 2) :=6 ,
MAP (2, 1) :=2 ,      MAP (2, 2) :=31,
MAP (3, 1) :=12,      MAP (3, 2) :=19,
MAP (5, 1) :=22,      MAP (5, 2) :=6 ,
MAP (4, 1) :=22,      MAP (4, 2) :=31,
(• PART 2...READ THE CHESS.DAT'CONTENT INTO AR ARRAY •)
3:
FIRST:=TRUE,
RESET (F, 'CHESS', 'DAT'),
FOR I:=1 TO 40 DO
BEGIN
FOR J:=1 TO 70 DO
IF NOT EOLN (F)
THEN READ (F, AR (I, J) )
ELSE AR (I,J) :=' ' ,
READLN (F)
END,
CLOSE (F),
(• PART 3...OUTPUT THE AR'S CONTENT TO TT:•)
2:
FOR I:=1 TO 22 DO
BEGIN
FOR J:=1 TO 70 DO WRITE (AR (I, J) ),
WRITELN
END,
IF NOT FIRST
THEN WRITELN('...A:', S:2, D:5, ' ':8, '...B:', FREE:2, S:5),
IF (A (1) +A (2) =6) AND ((FREE=1) OR (FREE=2))
THEN

```


BEGIN

WRITELN ('YOU ARE FAILED , FOR A MOMENT GO ON! '),

T:=TIME,

WHILE TTME-T<1/180 DO I:=I,

FOR I:=1 TO 2 DO

BEGIN

A (I) :=1, B (I) :=1+3

END,

FREE:=3,

GOTO 3

END,

(• PART 4...MATCH PROCESS •)

1:

WRITE('INPUT YOUR ANSWER:'),

READLN (S, D);

IF (S IN ST) AND (D IN ST)

THEN PEAPLE

ELSE GOTO 1,

COMPUTER;

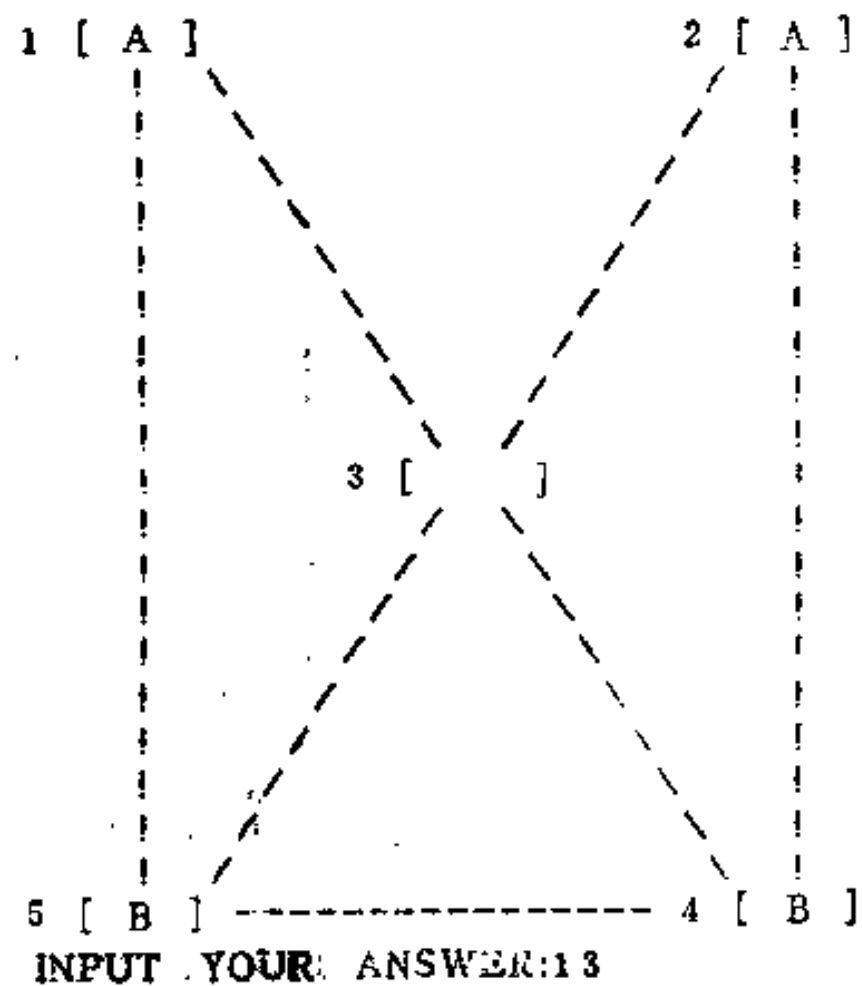
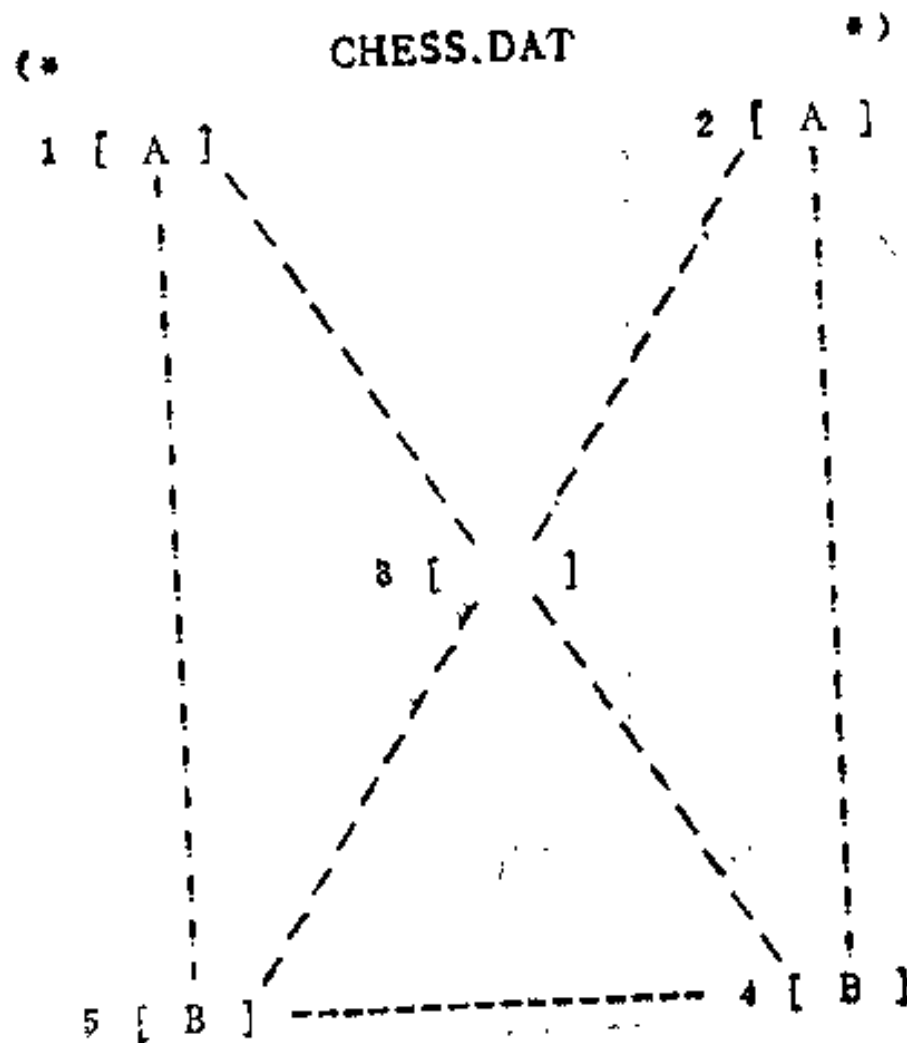
MODIFY;

FIRST:=FALSE,

GOTO 2.

END.

OUTPUT,



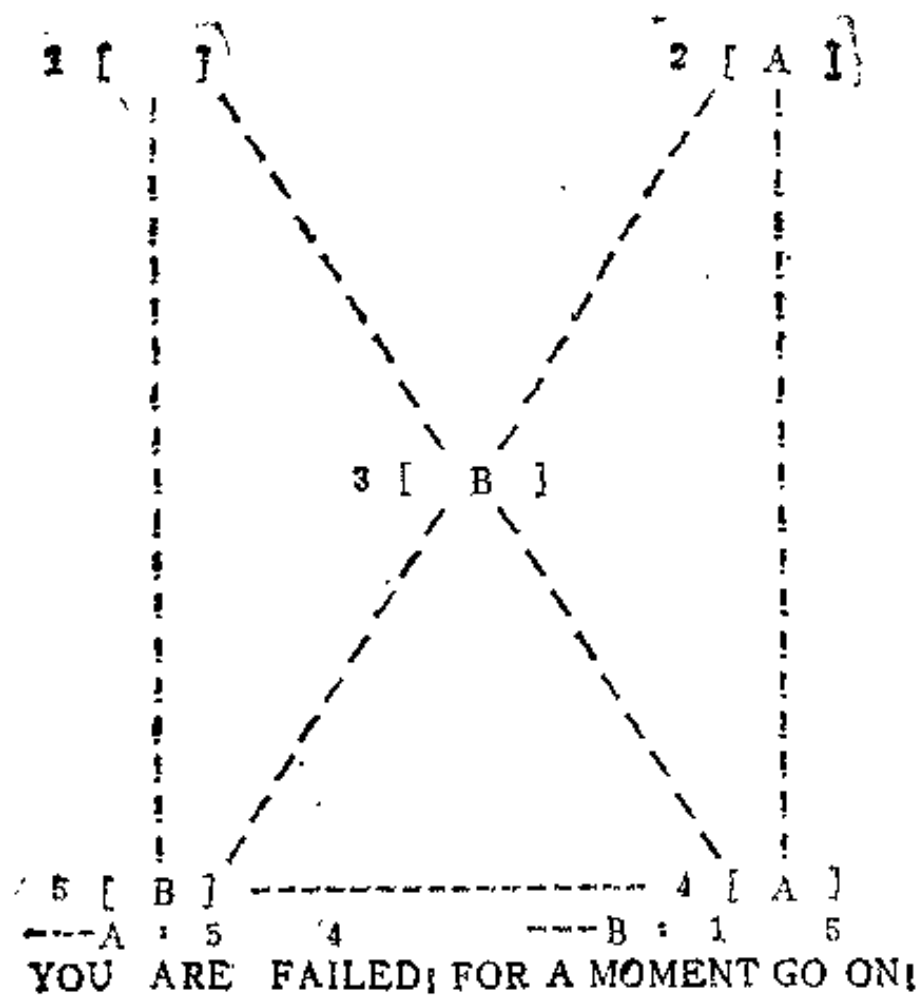
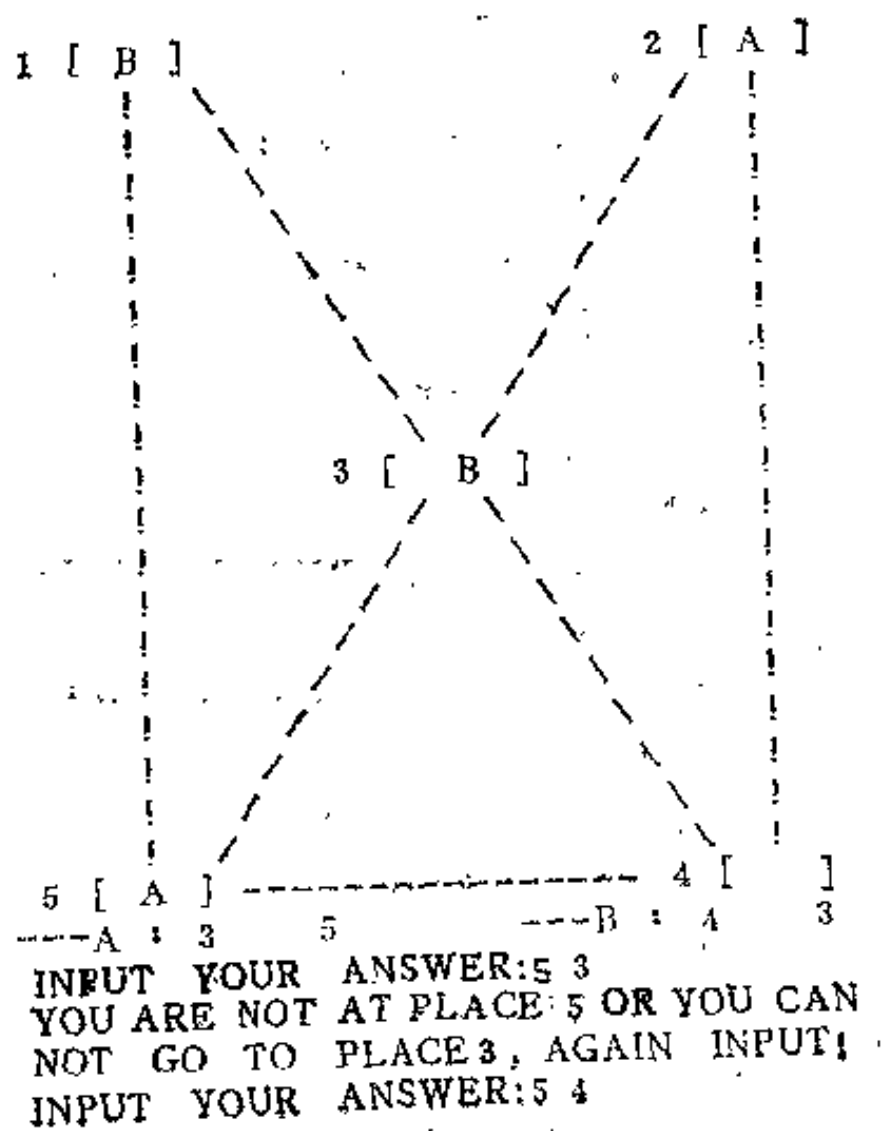
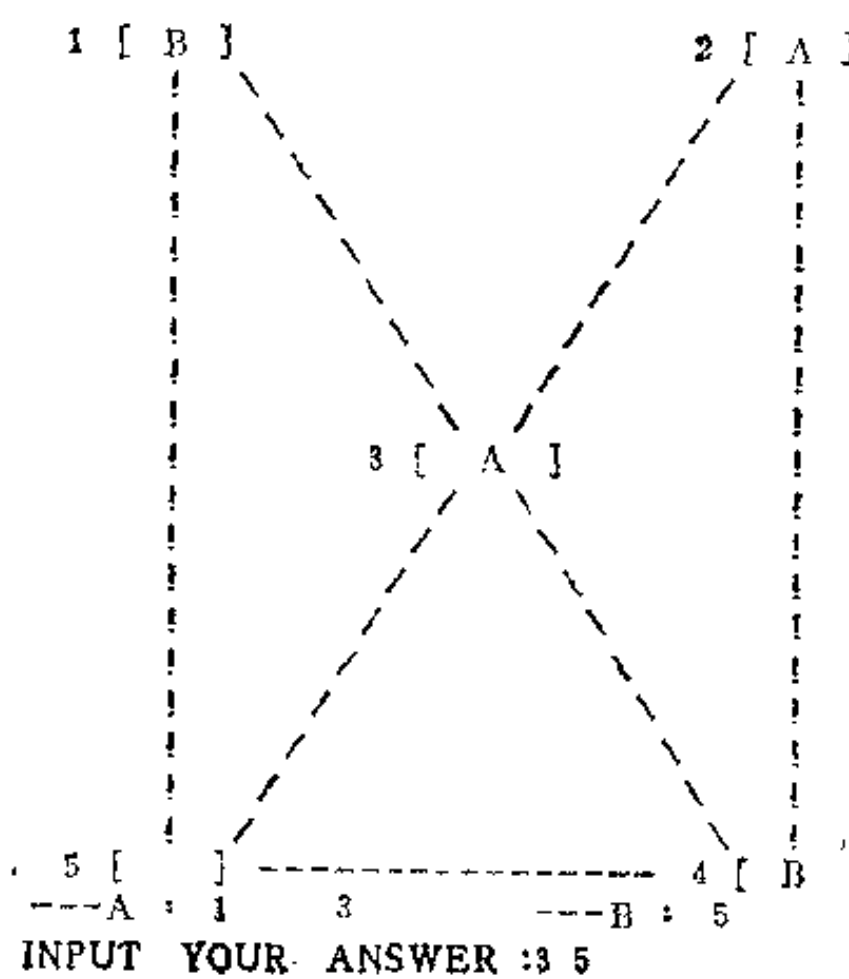
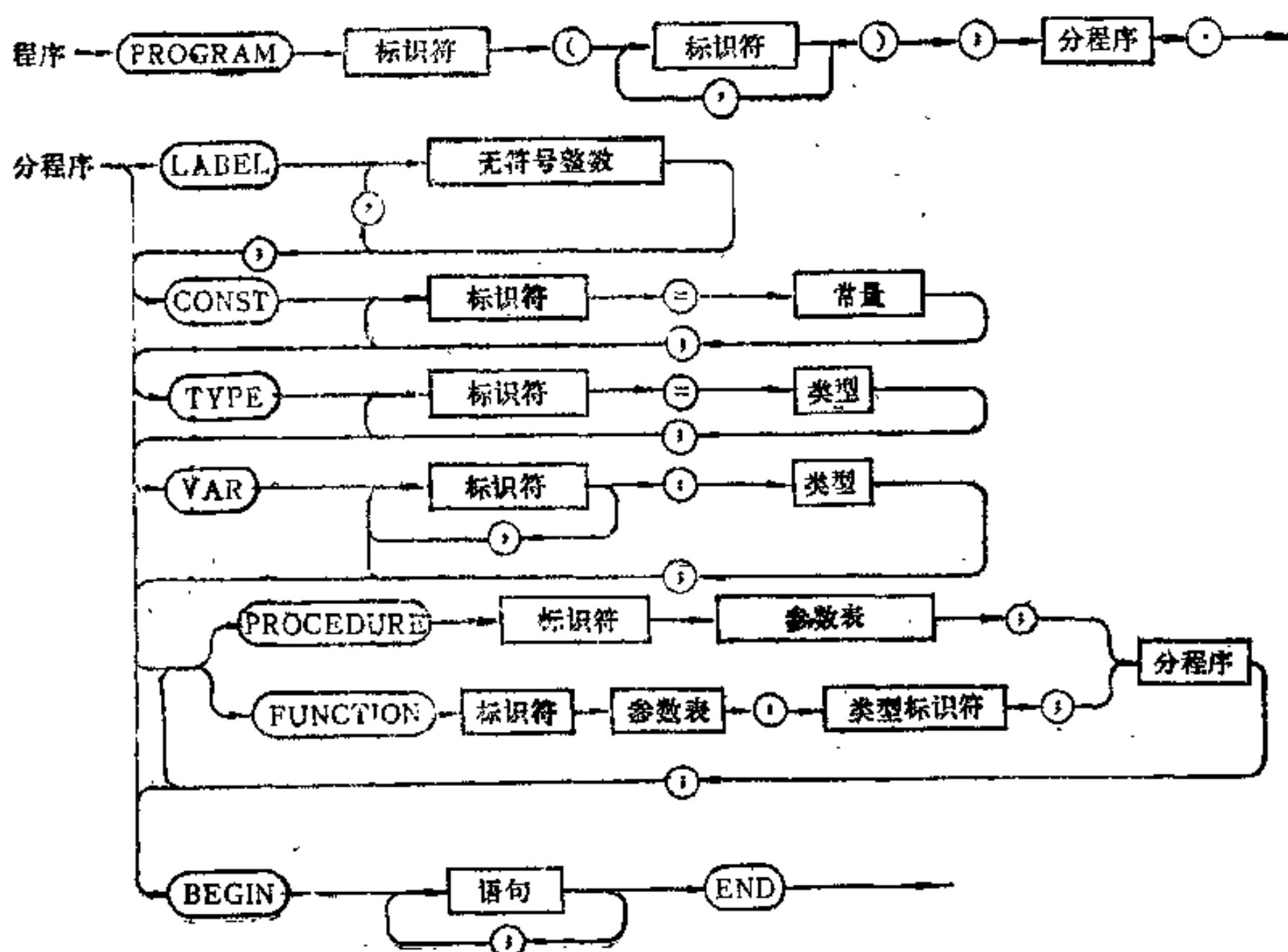
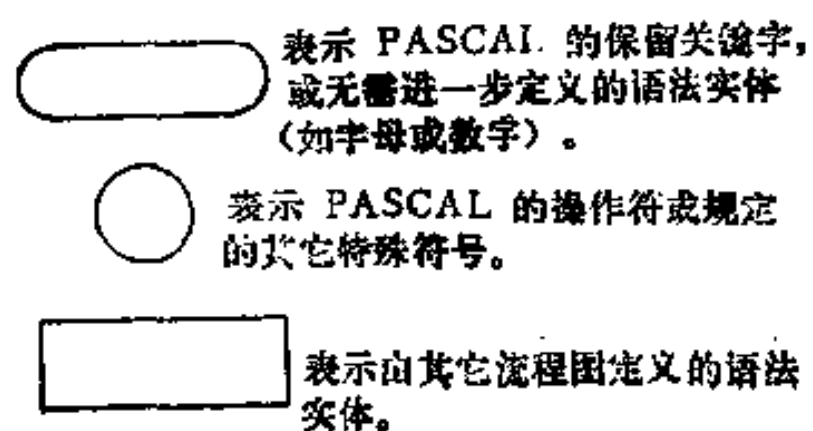


FIG TOTAL 12 PROGRAM

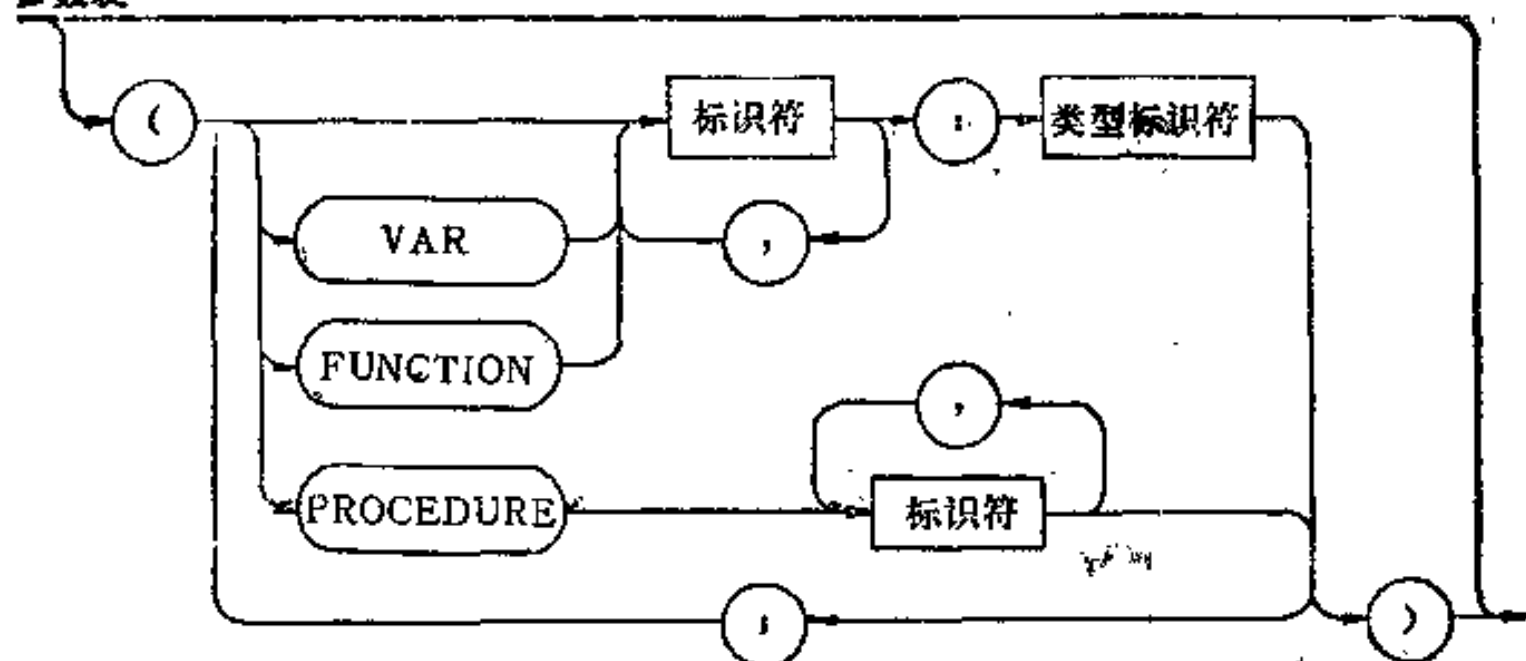
附录

附录一 PASCAL 语法图

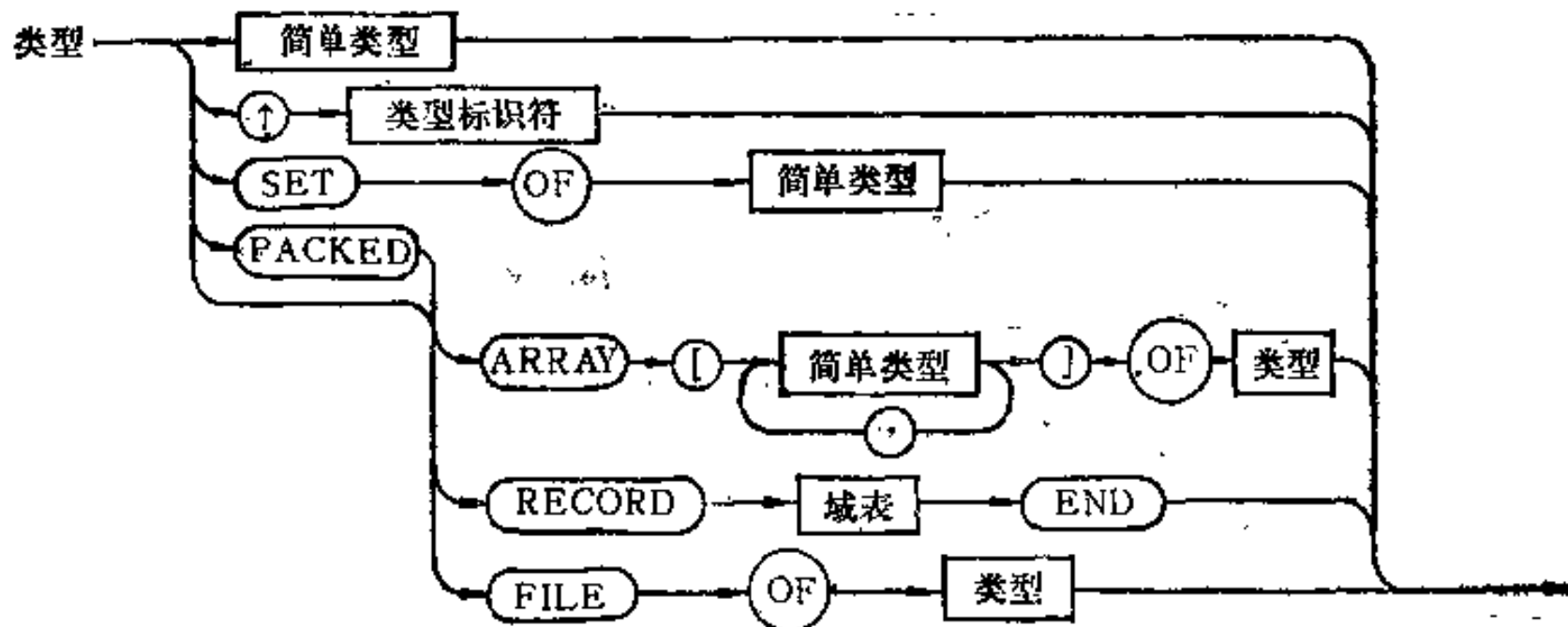
这一附录包括 PASCAL 的全部语法图。如果其它章节中的语法图与此不相符，则以此语法图为准。



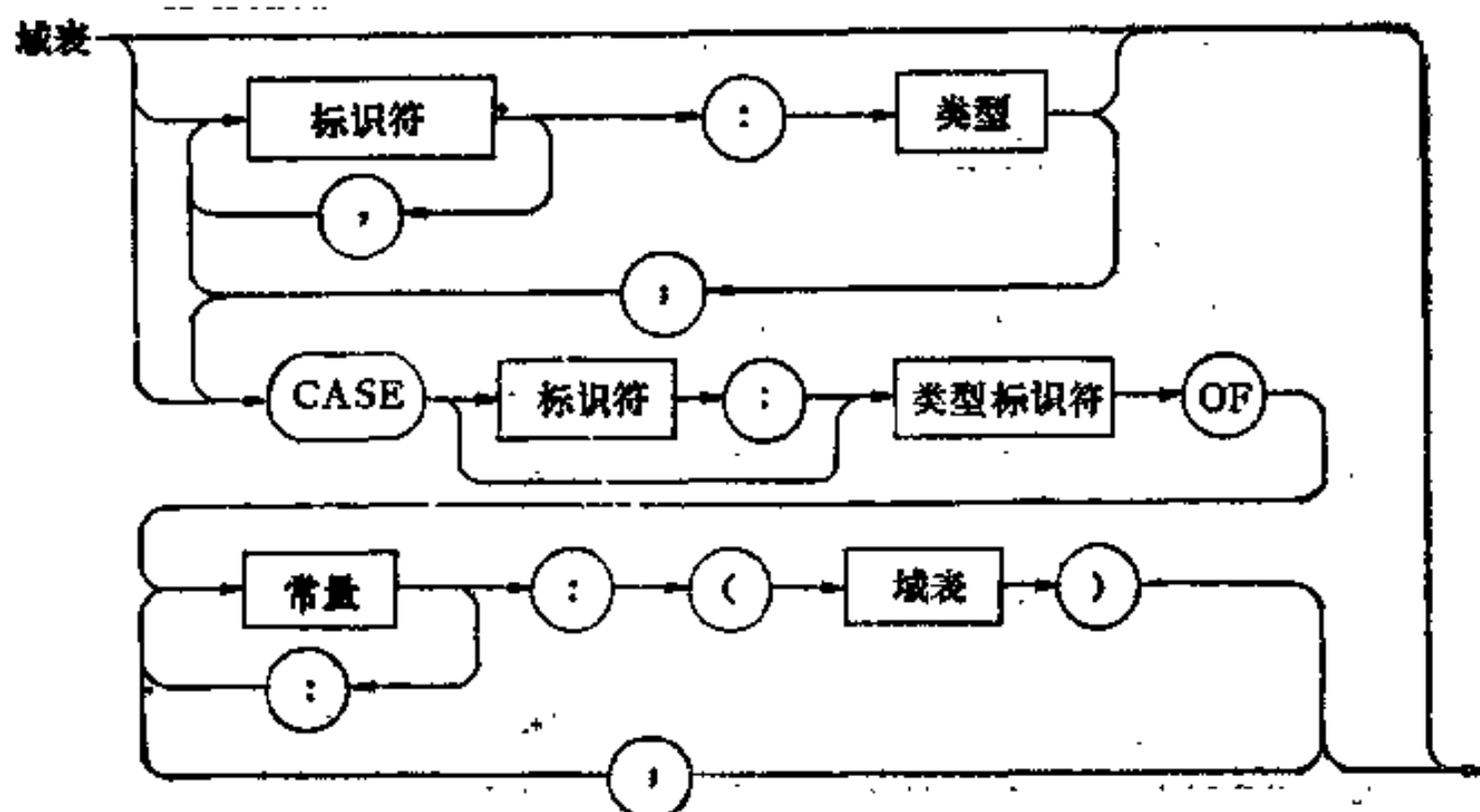
函数表



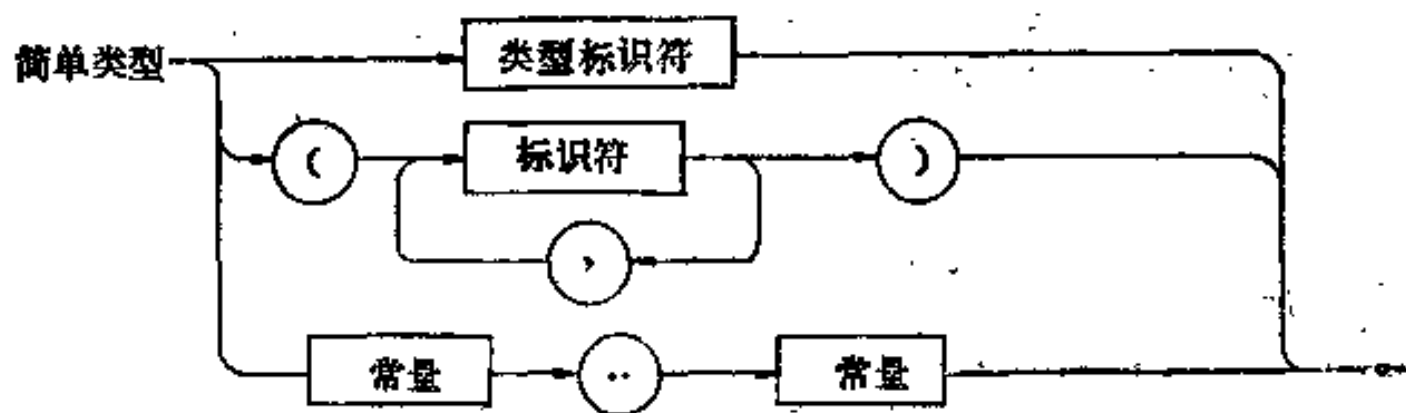
类型

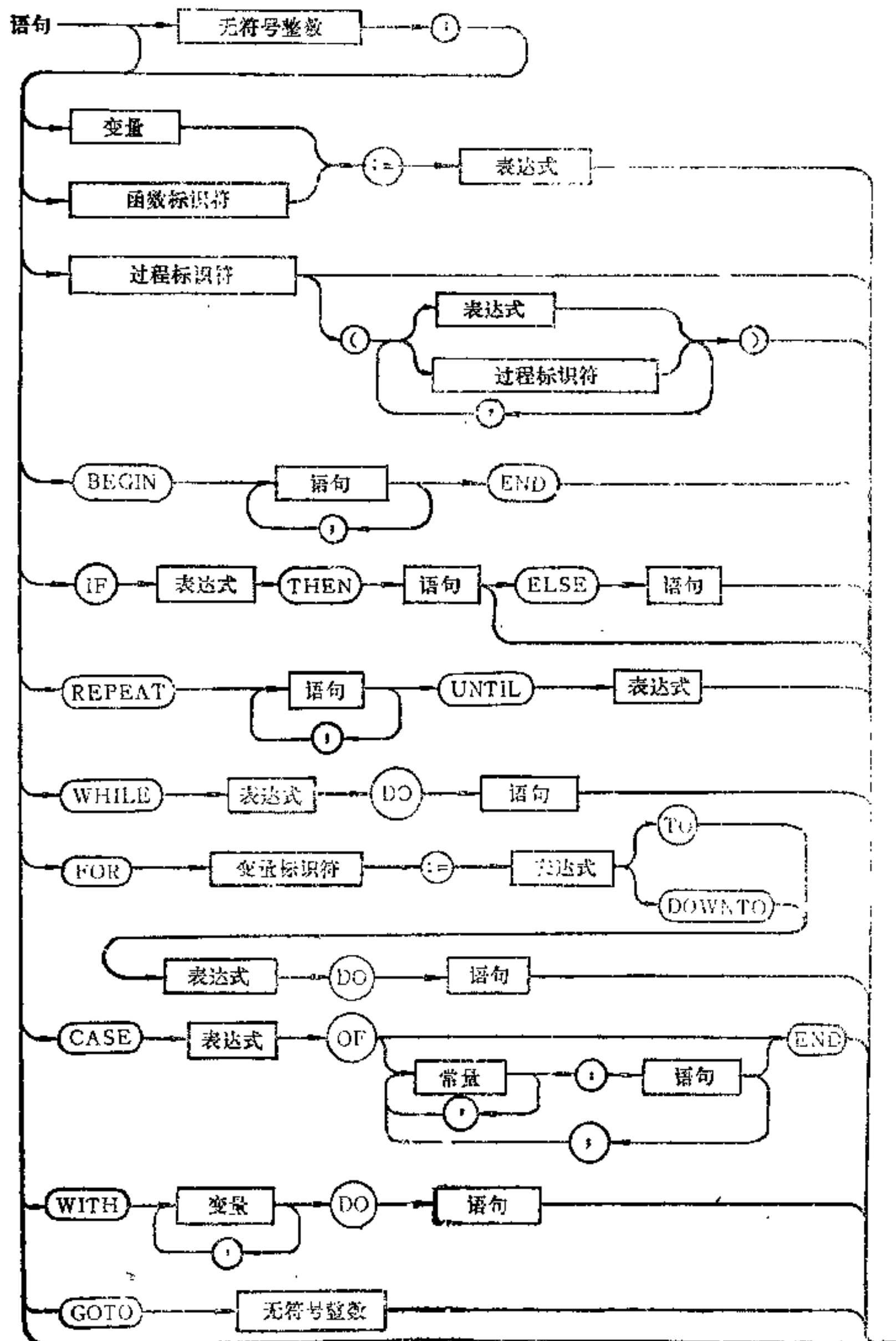


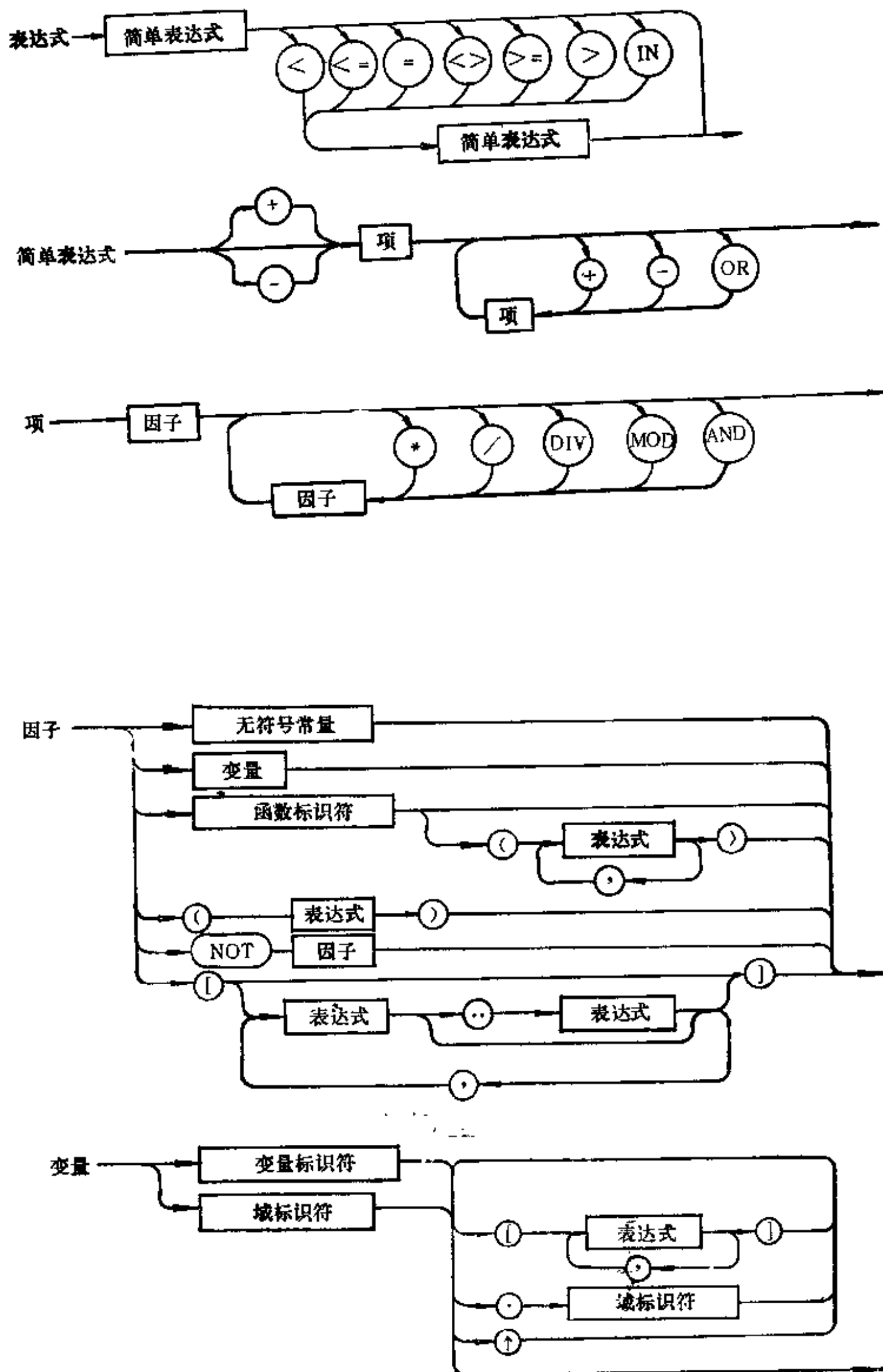
域表

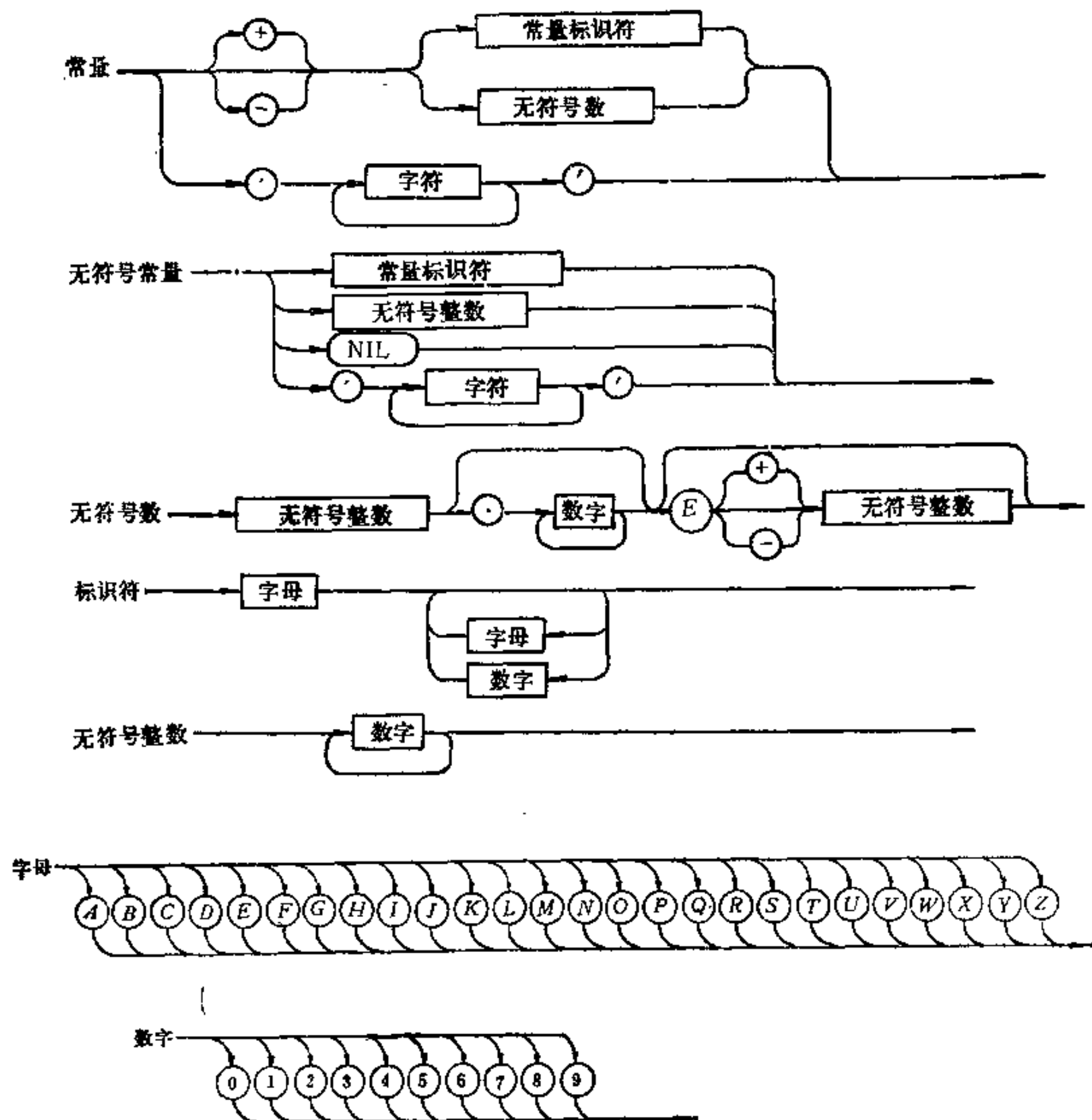


简单类型









附录二 保留关键字

在 PASCAL 语言中，下述保留关键字都有特定的意义，它们是作为不可分割的特殊符号处理的。除了在注解中以外，它们只能用于语言中定义的用途。

AND	ARRAY 数组	BEGIN	CASE 分情形	CONST 常量声明
DIV 带余除法	DO	DOWNTO 到	ELSE 否则	END
FILE	FOR 循环	FUNCTION 函数	GOTO 转到	IF
IN	LABEL 标号	MOD 取模运算	NIL 空	NOT
OF	OR 或	PACKED	PROCEDURE 过程	PROGRAM 程序
RECORD	REPEAT 重复	SET 集合	THEN 那么	TO
TYPE	UNTIL	VAR 变量	WHILE 当...时	WITH

附录三 标准标识符

在PASCAL语言中,用来标志标准的常量、类型、文件、过程和函数的标识符,叫标准标识符,也有叫标准关键字的。它们有特殊规定的意义,在程序中直接引用即可。

标准常量:

FALSE 真 MAXINT TRUE 真

标准类型: 字符类 整数类型 实数类型 文本类
BOOLEAN CHAR INTEGER REAL TEXT

标准文件:

INPUT OUTPUT

标准函数:

ABS 反切 ARCTAN 字符 CHR 文件结束 EOF 行结束 EOLN
EXP 指数 LN 自然对数 ODD 奇数 ORD 序 PRED 前导 ROUND 舍入取整
SIN SQR 平方值 SQRT 平方根 SUCC 后继 TRUNC 截尾

标准过程:

DISPOSE 取文件记录的过程 GET 建立数据 NEW 紧缩过程 PACK 建立新文件
PUT READ READLN RESET REWRITE
UNPACK WRITE WRITELN

附录四 ASCII 字符集

目前,在计算机程序设计语言中采用最广泛的是ASCII字符集。它是美国标准信息交换码 (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)。共有九十五个可打印的字符,字符码从32至126。其中字符码00至31以及127是控制字符,不能打印。字符码32表示的是空格字符。

ASCII CODE OF CHARACTERS

右侧数字 左侧数字	0	1	2	3	4	5	6	7	8	9
3				!	"	#	\$	%	&	'
4	()	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[\]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~			

附录五 OMSI PASCAL-1语言规范

在“PASCAL用户手册和报告”一书（作者为K.Jensen和N.Wirth）中，系统地定义了‘标准PASCAL’。

OMSI PASCAL-1对‘标准PASCAL’作了某些扩充和限制。本文对这些扩充和限制作简要的说明。

正确地使用在本文中阐述的扩充功能，可以用来编写某些程序，解决一些难以用标准PASCAL来处理的问题。但注意的是，用OMSI PASCAL-1编写的程序，不一定全部能在其它版本的PASCAL系统上运行。

§1. 语 法 扩 充

一、程序首部 在OMSI PASCAL-1中，程序首部是任选部分，可以省略不写。如果写了，程序名将被打印在程序清单每一页的开始位置。对程序首部中给出的所有程序参数均不予处理。

二、说明部分的次序安排 在OMSI PASCAL-1中，对总的说明部分（CONST, TYPE, VAR）的次序安排，作了新的扩充。它允许说明部分多次出现，只要做到在使用每个标识符之前，有关标识符已有定义即可。

应用这种办法，可以在主程序之外，建立另外一些过程或函数的模块，而在主程序中只要简单地把它们说明为外部过程或外部函数。这样，主程序就可以如同应用自己的库程序一样，应用那些过程或函数的模块。

三、注释括号 OMSI PASCAL-1允许三种格式的注释括号：

{.....}, (*.....*), /*.....*/

它们允许自由搭配。也就是说，开始的注释括号与结束的注释括号不一定具有相同的格式。

四、在CASE语句中的ELSE分语句 OMSI PASCAL-1允许一个ELSE分语句出现在CASE语句内部。

当CASE语句中的选择表达式的值，不属于该语句内的任何一种情况标号时，就转去执行ELSE分语句。ELSE分语句应跟在其它带标号的语句之后。

请注意，在ELSE之前、在其它带标号的语句之后的分号（;）不能省略。而分语句与结束情况语句的END之间，不得有分号（;）这个分隔符。

五、EXIT语句 EXIT语句用在各种重复语句（WHILE, REPEAT, FOR）里，当某种条件满足时，用EXIT语句强制退出尚未正常重复完的语句。

六、EXTERNAL和FORTRAN过程 OMSI PASCAL-1允许程序员使用与主程序分开编译的外部过程和外部函数。这就使程序员得以访问它们的程序库。为了在主程序中说明一个外部过程，要用关键字EXTERNAL。也可以用FORTRAN来代替EXTERNAL，但这时要求编译程序生成FORTRAN类型的调用。应该说明的是，FORTRAN调用时，所有过程参数必须都是引用调用，即所有参数都应是变量参数。

例如：

PROCEDURE ARASE; EXTERNAL;

FUNCTION DEFCE(VAR X, Y: REAL): REAL; FORTRAN;

§2. 低级接口

一、八进制常量 可以用八进制形式来表示整数型常量。任意一个整数型常量后紧跟字母'B'，表示这是一个八进制数。在数字与字母B之间不能有空格。当然，这时如果有数字8或9出现，则是一个错误。

例如：

CONST

TAB1=12B;

二、无符号整数 用在PDP-11上的表示算术运算的标准类型整数，范围是从-32768到+32767。也可以用0..65535这一子界来说明无符号整数。这时，编译程序将应用PDP-11的无符号运算部分，并且不处理无符号整数在乘、除运算过程中的溢出。

三、对整数进行逻辑运算 OMSI PASCAL-1已扩充到允许对整数进行与、或、非等逻辑运算。这种逻辑运算，是在两个16位的操作数之间每位上分别进行的。有了这一扩充，就可以对整数的每一个二进制位单独进行测试和修改。例如，对一个设备寄存器的各状态位进行检查。

四、访问绝对的存储器地址 有时，希望访问某些确定的存储器地址（也称绝对的存储器地址），如设备控制寄存器，或某些操作系统的特定参数表等。OMSI PASCAL-1使用关键字ORIGIN，在变量说明部分为某一变量指定一个特定的地址。

五、地址操作符@ OMSI PASCAL-1给出一个用字符@表示的单一操作数的地址操作符。当它被用于一个类型为'A'的变量时，得到的值为类型'↑A'（指向A的指针）。这一地址操作符，能够把一些变量勾链到它的列表结构中去，也常常把某些有关地址传送给相应的低级子程序。

六、插入宏汇编指令 OMSI PASCAL-1允许在程序的任何位置上，插入PDP-11的一段宏汇编指令。在这些插入的指令中，MACRO-11的各项特性均有效，而且编译程序还能给出访问PASCAL变量的方法。这就要求程序员必须了解PASCAL程序在运行时的情形。插入的宏指令，必须以特定的注释形式出现在PASCAL源程序中。例如：

BEGIN

...

(* \$ C

MOV n(6), -% 6

EMT 53

*)

...

END.

；开始插入MACRO指令

；把参数n存入堆栈

；调用EMT管理程序

；结束插入宏指令

请注意，只能用寄存器的编号来访问各寄存器（除非在插入的MACRO部分中重新定义R₀-R₅，SP和PC），并且，在PASCAL编译程序产生出来的MACRO文件中，如不特别说明，计算制用的是十进制，而不是八进制。

§3. I/O支持扩充

一、RESET和REWRITE过程中的补充参数 OMSI PASCAL-1对'RESET'和'REWRITE'两个过程作了扩充，允许用三个可选参数来指明内部文件与外部文件之间的关系。

```
PROCEDURE RESET(F:FILE; NAME, DEFEXT:STRING;
                VAR LEN:INTEGER);
```

这个 RESET 过程，把一个文件变量与一个现存的外部文件连通起来，并且把文件定位到它的第一个元素那里。‘F’是需要的，并且必须是一个文件变量。文件名 (NAME) 和文件类型 (DEFEXT) 是字符数组类型。它们必须符合于所用的操作系统对文件名的规定。如果没有明确给出文件的类型，则此时默认文件类型为‘DAT’。

参数‘LEN’，如果有的话，用于存放指定文件的长度。单位是 BLOCK(块)，每块由 512 个字节组成。如果要找的文件不存在，则 LEN 参数被置成 -1。必须说明的是，如果在一个 RESET 过程中，既没有给出参数 LEN，指出的又是一个不存在的文件，这是一个致命的程序错误。

```
PROCEDURE REWRITE(F:FILE; NAME, DEFEXT:STRING;
                  VAR LEN: INTEGER);
```

‘REWRITE’过程，用于建立一个新的外部文件。这里的‘NAME’和‘DEFEXT’参数的意义，与在 RESET 过程中的意义相同。而‘LEN’参数，指出的是已分配给这一文件的可用长度，单位仍是 BLOCK(块)。

二、BREAK 过程 为了提高系统效率，OMSI PASCAL-1 在向文件传送数据时，采用了缓冲技术。‘BREAK’过程的功能，是在缓冲区未满的情况下，强行把已存入缓冲区的数据传送走。在用到终端设备时，这一过程非常有用。例如，可以方便地向用户输出一行提示讯息。

三、CLOSE 过程 在程序处理完一个文件时，应调用‘CLOSE’过程。‘CLOSE’过程，把缓冲区中的数据传送给相应的文件，‘切断’与外部文件的联系，并释放内部缓冲区占用的存贮单元。在对文件变量进行操作之前，必须优先调用‘RESET’或‘REWRITE’过程。

对输出文件，‘CLOSE’过程是非有不可的。它使所建立的文件由暂时文件变为永久文件。而对输入文件，则可用可不用，如果用了，则释放了缓冲区占用的存贮单元。

四、读字符数组 已对标准的‘READ’过程进行了功能的扩充，使它允许读入任意长度的字符数组。每读一个字符数组时，总是从文件当前位置开始，顺序往下读，直到遇见空格、逗号或行结束标志时停止。读入的字符串，总是从字符数组的头一个元素（即下标最小的那个元素）算起，如果读入的字符个数比字符数组规定的长度短，不足部分填充空格，而当读入的字符个数多于数组长度时，多出的字符扔掉。

五、用 WRITE 过程输出八进制结果 如果需要，可以让‘WRITE’过程以八进制形式输出整数。这是通过对表达式给出用负值表示场宽来完成的。例如：

```
WRITE(1:-5);
```

表示用八进制形式输出变量‘1’，并让它占五个字符的位置。

六、随机存取文件 OMIS PASCAL-1 提供了一套过程定义，以允许对文件变量元素随机存取。

```
PROCEDURE SEEK(VAR F:RANFIL; VAR DATA:USERTYPE;
                N:INTEGER);
```

这个‘SEEK’（搜索）过程，能够在变量数据中搜索到文件 F 的第 N 个元素。如果这个元素不存在，那么就设置文件结束标志‘EOF(F)’为真。

```
PROCEDURE DOPOSIT(VAR F:RANFIL, DATA:USERTYPE,  
N:INTEGER);
```

这个'DOPCSIT' (写入) 过程, 就是把'DATA' (数据) 变量放进文件'F'的第N个元素位置上。有了这个过程, 如果必要的话, 可以扩充文件。

```
PROCEDURE CLOSERANDOMFILE(VAR F:RANFIL);
```

这个过程经常用来正确地关闭随机存取文件, 实际上就是代替'CLOSE'过程。

七、交互终端上的I/O问题 标准 PASCAL 语言规定, 对一个文件进行'RESET'操作之后, 文件的第一个元素便可得到并可以直接应用了 (即缓冲区变量 F ↑ 立即有确定的值)。

但在默认的输入文件为终端时, 将引起严重的困难。这是因为, 要满足上述要求, 用户必须在执行'RESET'语句之前, 就打进一个字符或打进一行字符, 这是很难办到的。

OMSI PASCAL-1是采用如下办法来解决这一问题的。在对交互作用文件 (终端) 进行'RESET'操作时, 在实际的 I/O 传送并未开始的情况下, 就直接设置缓冲区变量为空格 (SPACE), 并使'EOLN'为真。

每一次读请求, 都应读进相应个数的数据, 在未满足它需要的足够的数据之前, 它等待输入, 直到足够为止。当然也不会读入更多的数据。

用这种方法, 解决了应用终端输入时遇到的大多数问题, 但同时带来了另外一个困难。

在应用终端输入时, 这一方法不能辨认一个空行和仅包括一个空格的一行。这是因为, 不能在打完一行字符之前就设置'EOLN'为真, 因此, 无法满足'READ'的请求。

§4. 补充说明的一些过程

一、DISPOSE 过程 在标准 PASCAL 中, 已删除'DISPOSE'过程, 但在 OMIS PASCAL-1中仍保留了它。

从理论上讲, 通过'DISPOSE'过程释放了的存储空间, 可以被'NEW'过程再次利用。但应用中的一些实际问题, 留给用户自己去处理。

必须指出, 在调用'NEW'过程时, OMIS PASCAL-1总是按一个记录的最大长度来分配存储单元。因此, 出现在'NEW'和'DISPOSE'过程中的记录变体标志, 对分配与释放存储单元毫无影响。

二、TIME 函数 'TIME' (时间) 函数, 以实数形式送回 24 小时制表示的本日的当前时间。如上午九点三十分表示为 9.5, 下午一点四十五分表示为 13.75。时间的精度取决于所使用的操作系统, 一般情况为一秒, 或者更小。

三、EXP10 和 LOG 函数 'EXP10'和'LOG'函数, 与标准的'EXP'和'LN'函数是类似的, 所不同的是它们的对数底为 10。

§5. 非标准的语言元素

一、PACKED 变量 从有效地管理紧缩 (PACKED) 结构数据 的角度出发, 标准 PASCAL 定义了'PACK'与'UNPACK'过程。OMSI PASCAL-1不处理这种压缩 (PACKED) 属性 (在标准 PASCAL 中是允许的), 并且不允许使用'PACK'和'UNPACK'两个过程。

二、READ 和 WRITE 过程中的默认文件 标准 PASCAL 定义了两个字符文件变量 'INPUT'和'OUTPUT', 它们分别表示'READ'和'WRITE'过程中的默认文件 (具体地说, 'READ(I)'与'READ(INPUT, I)'等同, 'WRITE(I)'与'WRITE(OUTPUT, I)'等同) 然而 OMIS PASCAL-1不再预先说明'INPUT'和'OUTPUT'两个文件, 而把

用户的终端作为‘READ’和‘WRITE’两个过程的默认文件。

三、输出文件必须用 CLOSE 过程加以关闭 OMSI PASCAL-1 要求，在处理完一个输出文件时，必须用‘CLOSE’过程来关闭它。否则，很可能导致丢失存放在缓冲区中的最后一批输出数据。

四、过程和函数参数 通常，标准的（或预先说明的）过程和函数，不能作为参数用在其它的过程参数中。但如果确有此需要，可以再说明一个过程，在这个过程中只简单地调用标准的过程就行了。例如为了使用标准函数 SIN，

```
FUNCTION SINE (X:REAL) :REAL;  
BEGIN  
    SINE:=SIN (X)  
END;
```

五、文件的文件 OMSI PASCAL-1 不允许使用文件作为另一个文件的元素，但允许把文件作为数组的元素和记录的一个域。

六、文件的结束问题（仅适用于RT11, RSTS） 在 RT11 和RSTS 操作系统中，文件是用若干个由 512 个字节组成的块（BLOCK）组成的。没有一种很好的办法，在最后一块中用来准确地表示数据的结束。

因此，不能用‘EOF’这个标准函数来判断文件的结束。这时，可用其它办法来解决这一问题，例如可以通过计算文件元素的个数，或查出某一特定的文件元素来判定文件的结束。

请注意，上述说明不适用于 TEXT 文件中，因为 TEXT 文件是用字符 CONTROL/Z 来标记文件结束的。

§6. 实现编译程序时的某些规定

一、标识符 OMSI PASCAL-1 允许标识符有任意长度，并且全部字符都有效。小写体字母也可以用，并将变换为大写体字母（如‘fee’=‘FEE’）。但标准 PASCAL 建议，所有标识符的前八个字符应有差异，不能完全相同，以便能相互区别。

由于目标程序文件结构上的限制，外部过程以及 FORTRAN 过程的标识符的前六个字符，必须是单值的，即彼此不同。

二、标准类型 INTEGER 在 PDP-11 系统中，标准类型 INTEGER 的范围是(-32768..+32767)。无符号整数可以用0..65535 这一子界来说明。对于无符号整数的乘、除运算，不进行算术溢出检查。

‘MAXINT’是预先说明的一个常量名，可以直接出现在用户程序中，它的值为32767。

三、标准类型 REAL OMSI PASCAL-1 利用标准的 PDP-11 单精度浮点结构表示实数，其范围约在 $1E-38..1E+38$ ，精度是十进制 7 位数字。对所有实数运算都有溢出检查，下溢时只给出零值结果，不另行处理。

四、ASCII 字符集 OMSI PASCAL-1 采用 7-bit（七位二进制数）表示的全部 ASCII 字符集。字符将以正负号字节（即 8-bit）形式存放。因此 ORD (CHAR) 的范围为(-128..127)

五、集合的元素个数的限制 PASCAL 报告一文中指出，为提高系统运行效率，应该把集合的元素个数限制在一个比较小的值内。OMSI PASCAL-1 规定，一个集合最多可以有 64 个元素。

对标准的字符类型，集合的默认范围是 ASCII 码表中从空格字符开始到小写字母之前

的可打印的64个字符。

六、过程嵌套的最大深度

过程的最大嵌套深度规定为10层。

对递归的实际深度未作任何限制，但是递归的次数太多，可能用尽全部内存单元，使程序无法继续运行。

七、WITH 语句的最大嵌套深度 在记录类型的 WITH 语句，最大嵌套深度为四层。如果在 WITH 语句范围内，用到了比较复杂的表达式，编译程序会把这一嵌套深度限制为二层或三层。

附录六 OMSI PASCAL-1程序编译及运行的错误信息表

OMSI PASCAL-1 程序编译错误信息一览表

ALL CHARACTERS IGNORED UNTIL...	在...之前的所有字符均不予处理
ALL VAR DEFINITIONS PROCEED PROCEDURE DEFINITIONS	全部变量应在过程定义前予以说明
ARGUMENT MUST BE INTEGER	参数必须为整数
ARGUMENT MUST BE INTEGER TYPE	参数必须为整数型
ARGUMENT MUST BE NON-REAL SCALAR	参数必须为非实数型的纯量类型
ARGUMENT MUST BE REAL	参数必须为实数
ARRAY INDEX OUT OF RANGE	数组下标超出范围
ARRAY INDEX TYPE ERROR	数组下标的类型错
ASSUMED WHERE INDICATED	假定这里已指出
BAD 'ABS' ARG	求绝对值中的参数有错
BAD ARGUMENT	参数有错
BAD CASE LABEL	CASE语句标号有错
BAD CONSTANT	常量错
BAD 'EXIT'	EXIT 用错
BAD EXPRESSION	表达式有错
BAD FILE NAME	文件名有错
BAD FIELD LIST	域表有错
BAD FOR STATEMET	FOR 语句有错
BAD FUNCTION NAME	函数名有错
BAD FUNCTION RESULT TYPE	函数结果类型有错
BAD 'IN' OPERANDS	IN 的操作数有错
BAD INDEX TYPE	下标类型错
BAD INTEGER	整数有错

BAD LABEL	标号错
BAD ORIGIN FOR VARIABLE	为变量指定的绝对地址有错
BAD PARAMETER	参数不对
BAD PROCEDURE NAME	过程名有错
BAD PROGRAM NAME	程序名有错
BAD READ STATEMENT	读语句有错
BAD RECORD	记录有错
BAD SCALAR TYPE	枚举类型有错
BAD SET ELEMENT	集合的元素有错
BAD SUBRANGE	子界有错
BAD TYPE	类型错
BAD TYPE SPECIFICATION	类型说明有错
BAD VARIABLE LIST	变量列表有错
BAD VARIANT	变体有错
BAD WITH STATEMENT	WITH语句有错
BAD WRITE STATEMENT	写语句有错
BOOLEAN EXPRESSION NEEDED	应该用布尔表达式
CONSTANT OVERFLOW	常量溢出
DUPLICATE CASE LABEL	CASE语句中标号用重了
DUPLICATE FIELD NAME	域名用重了
ELSE NOT LAST IN CASE STMT	ELSE未处在CASE语句最后
EXPECTED',.'	期待',.'
FATAL ERROR:	严重错误:
FIELD LIST MUST BE IN PARENTHESIS	域表清单必须在括号中
FILE VARIABLE MISSING	文件变量遗漏
FOR STATEMENT MUST BE INTEGER TYPE	FOR语句必须是整型变量
FORTRAN PARAM. MUST BE CALL BY REFERENCE	FORTRAN过程参数必须根据规定 标准调用
FUNCTION ARG MUST BE REAL OR INTEGER	函数的参数必须为整数或实数
FWD. PROC. DON'T REPEAT PARAMETER LIST	提前说明的过程不必重复参数表
ILLEGAL ASSIGNMENT	非法赋值语句
ILLEGAL OPERATOR	非法操作符
ILLEGAL TYPE OF OPERAND	操作数的类型非法
IMPROPER SYMBOL	不应出现的符号
INCOMPATIBLE ARRAY	数组不相容
INCOMPATIBLE TYPE	类型不相容

INPUT FILE TOO LONG
 INVALID CHARACTER 8 OR 9
 IN OCTAL CONSTANT
 INVALID DECLARATION
 INVALID SYMBOL
 LABEL DEFINE AT WRONG
 LEVEL
 LABELS MUST BE INTEGERS
 LABEL NOT DECLARED
 LABEL REDEFINITION
 MISSING
 MUST BE SIMPLE VARIABLE
 MISSING BEGIN
 MISSING END
 MISSING END IN CASE STMT
 MISSING FIELD VARIABLE
 MISSING LABEL
 MISSING LABEL DEFINITION
 MISSING OPERAND
 MISSING OPERATOR
 MISSING SEMI-COLON
 MISSING UNTIL
 MISSING ' .' AT PROGRAM END
 MISSING ') '
 MISSING ') ' AT END OF
 PARAM LIST
 NEW OR DISPOSE ARG
 NOT POINTER TYPE
 NO IMPLEMENTED
 'ODD' APPLIED TO NON-
 INTEGER EXPRESSION
 OUT OF FLOATING AC'S
 OUT OF REGISTERS
 PROBABLY MISSING END
 STRANGE '{' -- BAD SET
 OR MISSING ARRAY DEF
 TOO FEW ARGUMENT
 TOO MANY ARGUMENTS
 TOO MANY ERRORS IN THIS
 LINE

输入文件太长
 在八进制常量中8或9属非法字符

 这样说明不合法
 非法符号
 标号定义在不恰当的层中

 标号必须为整数
 标号未加说明
 标号定义多次
 遗漏
 必须为简单变量
 遗漏BEGIN
 遗漏END
 在CASE语句中遗漏END
 遗漏域变量
 遗漏了标号
 遗漏标号定义
 遗漏操作数
 遗漏操作符
 遗漏分号
 遗漏UNTIL
 在程序最后遗漏'.'
 遗漏')'
 在参数表之后遗漏')'

 用在 NEW 或DISPOSE 中的参数
 不是指针类型
 未予实现
 不能对非整型表达式判断结果是否为奇数

 浮点累加器溢出
 寄存器溢出
 可能遗漏了END
 符号'{'表示集合有错或遗漏数
 组定义
 参数太少
 参数太多
 在这行错误太多

TOO MANY LEVELS	分层太多
TOO MANY SYMBOLS	符号太多
TOO MANY WARNINGS	警告性错误太多
UNDEFINED FORWARD PROCEDURE OR FUNCTION	未定义提前说明的过程或函数
UNRESOLVED FORWARD TYPE REFERENCE	前面的类型标记符号未定
UNDEFINED OPERAND -- ASSUMING INTEGER	操作数未定义, 假定它为整数
UNDEFINED POINTER BASE TYPE	未定义指针的基类型
UNDEFINED SYMBOL	未定义的符号
WITH IN REG 0	陷入 0 (零) 寄存器
',' USED INSTEAD OF ','	错用 ',' 代替了 ';'
ZSK OVERFLOW	磁盘上内容太多, 写不下

OMSI PASCAL-1 程序运行错误信息一览表

Array bounds error 数组越界错误, 表示企图访问不存在的数组元素。

Bad Set expression 集合表达式错, 表示集合的元素不在规定范围之内。

Divide by zero 零做除数

END of file on device

文件已结束, 表示 EOF (F) 为真时还调用 READ 或 GET 语句。

Floating point error

浮点错, 表示浮点实数计算上溢或下溢, 或负数取对数, 或指数函数溢出, 或负数求平方根。

Integer error

整数错误。表示截尾函数 (TRUNC), 舍入函数 (ROUND) 或其它运算时整数溢出。

I/O Channel not open

输入/输出通道没有打开, 表示对关闭的文件进行操作 (文件没有 RESET 或 REWRITE)。

No room on device or file not found

设备上没有足够位置或文件找不到。表示调用 RESET 过程没找到所指定的文件, 或调用 REWRITE 过程不允许建立新文件。

Not a valid device

不是一个有效的设备。表示 RESET 或 REWRITE 语句指定了非法设备。

Not enough available memory

没有足够可用的存储单元, 表示内存可用存储单元已用完。这可能是由过程 NEW 或过狂的嵌套、递归的使用不当造成的。

附录七 中英用语对照表

为了方便读者学习有关 PASCAL 程序设计方面的英文科技书藉和报告文摘, 对本书中用到的专用词汇, 编成以下中英用语对照表, 供参考。

一致符号	一 划 unique symbol
一遍编译程序	one-pass compiler
二叉树	二 划 binary tree
入口代码	entry code
下标	三 划 index
下标计算	indexing
下标变量	indexed variable
下标类型	index type
下界	lower bound
下溢	downflow
工具	facilities
上下文	context
上界	upper bound
上溢 (溢出)	overflow
大小	size
子界	subrange
子界类型	Subrange types
中央处理机	四 划 central processing unit (CPU)
无标号语句	unlabelled statement
无符号实数	unsigned real
无符号常量	unsigned constant
无符号数	unsigned number
无符号整数	unsigned integer
区分符 (分类符)	specifier
分号	semicolon
分析程序	analyser, parser
分界符	delimiter
分配存贮单元	allocation
分情况语句 (情况语句)	case statement
分情况标号 (情况语句标号)	case label
分程序	block

分隔符	separator
文本文件类型	TEXT
文字	literal
文件	file
文件结束	end of file
文件结束函数	EOF (F)
文件变量	file variable
文件类型	file type
文件清单	file list
文件缓冲区	file buffer
内部的 (内在的)	built-in, internal
内部过程	internal procedure
内部函数	internal function
比例因子	scale factor
反正切函数	ARCTAN (x)
五 划	
主程序	main program
正负号	sign
正弦函数	SIN (x)
出口代码	exit code
出错信息	error message
记录	record
记录变量	record variable
记录段	record section
记录类型	record type
记法 (标志)	notation
长度	length
生成程序	generator
对象 (目标)	object
对象表	object table
加法	add, addition
加法运算符	adding operator
打印	print
打印机	printer
可执行的	executable
布尔 (布尔类型)	Boolean
写	write
写语句 (输出语句)	write statement
写过程	WRITE (...), WRITELN (...)
外部的	external

外部过程
外部函数
目录
平方值函数
平方根函数
汇编
汇编程序
用户自定义的枚举型数据

external procedure
external function
directory
SQR (x)
SQRT (x)
assemble
assembler, ASEMBL
user-define scalar data

六 划

向前引用
后继
后继函数
访问 (存取)
设备
因子
关系运算符
优先顺序
动作
动态存储分配
安排
字分界符
字汇表
字母
字母或数字
字符
字符号
字符类型标识符
字符函数
字符集
过程
过程参数
过程标识符
过程说明
过程语句
过程首部
当语句
行文文件
行结束
行结束函数
多入出程序

forward reference
successor
SUCC (x)
access
device
factor
relational operator
precedence
action
dynamic storage allocation
layout
word-delimiter
vocabulary
letter
letter or digit
character
word symbol
CHAR
CHR (x)
character set
procedure
procedure parameter
procedure identifier
procedure declaration
procedure statement
procedure heading
while statement
textfile
end of line
EOLN (F)
I/O-bound program

多重循环语句	multiple for statement
多维数组	multidimensional array
如果语句	if statement
全程变量	global variable
列表	Listing
成分	component
成分变量	component variable
成分类型	component type
有序集	ordered set
有穷图	finite graph
自底向上	bottom-up
自顶向下	top-down
自然对数函数	LN (x)
七 划	
串	string
系统	system
系统文件	system file
系数	coefficient
注释	comment
体制 (组织)	organization
位串	bitstring
位移量	displacement
序号	ordinal number
序号函数	ORD (x)
进退函数	further function
返回	return
形式参数	formal parameter
形式的	formal
条件语句	conditional statement
层 (层号)	level
局部	local
局部变量	local variable
纯量	scalar
纯量类型 (枚举类型)	scalar type
状态	state, status
初始信息	source information
初值	initial value
余弦函数	COS (x)
应用	application
应用程序	application program

应用软件	application software
应用举例	application example
间接递归	indirect recursion
宏	macro
宏汇编 (宏汇编程序)	MACRO
运行	run
删除	delete
改名	rename
建立动态变量过程	NEW (p)
建立新文件过程	REWRITE (F)
八 划	
表	list, table
表达式	expression
变元 (参量)	argument
变体	variant
变体部分	variant part
变量	variable
变量参数	variable parameter
变量说明	variable declaration
变量说明部分	variable declaration part
变量标识符	variable identifier
语句	statement
语句部分	statement part
语法图	syntax graph
语法实体	syntactic entity
直到语句	repeat statement
直接递归	direct recursion
函数	function
函数引用	function reference
函数参数	function parameter
函数说明	function declaration
函数首部	function heading
函数标识符	function identifier
命令	command
命名符 (标志符)	designator
并 (并运算)	union
参数组	parameter group
取指令	fetch instruction
取消动态数据过程	DISPOSE (P)
实际的 (实在的)	actual

实际参数	actual parameter
实体	entity
实现	implementation
实数	real
实数类型	real type
实数类型标识符	REAL
定义	definition
定义域 (定义范围)	domain
定点表示	fixed-point representation
空	empty
空白符	blank
空格	space
空语句	empty statement
空集合	empty set
空操作	no-op
固定部分	fixed part
图式 (方案)	schema
终值	final value
松展 (展开)	unpack
松展过程	UNPACK (Z, A, I)
构造成分 (组成部分)	constituent
构造型语句	structured statement
构造类型	structured type
检查	inspect
枚举类型	enumerated type
转移语句	goto statement
转换函数	transfer function
奇数函数	ODD (x)
取文件记录的过程	GET (F)
取模运算符	MOD

九 划

重复语句	repetitive statement
复合语句	compound statement
复原过程	RESET (F)
复杂类型	complex type
复制 (拷贝)	copy
复数	complex number
说明	declaration
说明书 (规范)	specification
前缀	Prefix

首部	head, heading
送回	return
保留	reserve
保留关键字	reserved word
总体结构	architecture
指针	pointer
指针变量	pointer variable
指针类型	pointer type
指挥命令 (伪指令)	directive
指数函数	EXP (X)
结果类型	result type
结构 (配置)	structure, configuration
绝对代码	absolute code
绝对地址	absolute address
绝对值函数	ABS (X)
退出语句	EXIT
标记	master
标号	label
标号语句	label statement
标号说明部分	label declaration
标识符	identifier
标志域	tag field
标准基本类型	standard basic type
标准输入文件	INPUT
标准输出文件	OUTPUT
相对位移量 (变位)	displacement
显示	display
段 (节)	segment, section
项	term
顺序的	sequential
顺序访问结构	sequential access structure
类型	type
类型定义	type definition
类型标识符	type identifier
类型说明	type specification
除法运算	division
前导函数	PRED (X)
	十 别
乘法运算	multiplication
乘法运算符	multiplaying operator

调用	call
辅助调试程序	DEBUGGER
读	read
读语句	read statement
读过程	READ (...), READLN (...)
递归	recursion
递归子程序	recursive descent
特殊符号	special symbol
预定义	predefine
紧缩 (压缩)	pack (ed)
紧缩过程	PACK (A, 1, Z)
紧缩数组	packed array
陷阱	trap
素数 (质数)	prime
校正	correct
浮点表示	floating-point representation
矩阵	matrix
添加	append
添写记录到文件的过程	PUT (F)
随机访问结构	random-access structure
逻辑“与”	logical 'and'
逻辑记录	logical record
逻辑“或”	logical 'or'
逻辑真值	true value
控制变量	control variable
接口	interfacing
域	field
域表	field list
域标识符	field identifier
基本符号	basic symbol
基址寄存器	base register
基类型	base type
基数	cardinal
常量	constant
常量定义	constant definition
常量标识符	constant identifier
堆栈	stack
盘区 (块)	block
舍入函数	ROUND (X)

十一 划

append
 PUT (F)
 random-access structure
 logical 'and'
 logical record
 logical 'or'
 true value
 control variable
 interfacing
 field
 field list
 field identifier
 basic symbol
 base register
 base type
 cardinal
 constant
 constant definition
 constant identifier
 stack
 block
 ROUND (X)

遍历
窗口
循环表
循环语句
编址
编译
编译程序例行程序
编辑 (编辑程序)
替代
赋值语句
链
链表
链接 (链接程序)
联机
联机调试程序
联结编辑程序
程序
程序地址寄存器
程序状态寄存器
程序参数表
程序首部
程序部分
程序流程图
程序库
嵌套
集合
集合并
集合交
集合差
集合类型
属性

源代码
源行文
源程序
输入设备
输出设备
数字
数字序列

十二 划

traverse
window
for list
for statement
addressing
compile
compiler routine
EDIT
substitute
assignment statement
chain
chained list
LINK
on-line
ODT
linkage editor
program
program address register
program status register
program parameters
program heading
program part
flowchart
library
nest
set
set union
set intersection
set difference
set type
attribute

十三 划

source-code
source text
source program
input device
output device
digit
digit sequence

数组	array
数组变量	array variable
数组类型	array type
数组越界错误	array bounds error
数值参数	value parameter
数据结构	data structure
简单表达式	simple expression
简单语句	simple statement
简单类型	simple type
叠印	overprint
溢出	overflow
模式识别	pattern recognition
模块	module
辖域	scope
截尾函数	TRUNC (X)
整体变量	entire variable
整数	integer
整数变量	integer variable
整数类型标识符	INTEGER
整数除运算符	DIV
蕴含 (蕴涵)	implication, contain
静态的	static
默认的	default
默认文件	default file
默认设备	default device
n 维数组	n-dimensional array

十四 划

pattern recognition
module
scope
TRUNC (X)

十五 划

entire variable
integer
integer variable
INTEGER
DIV
implication, contain

十六 划

static
default
default file
default device
n-dimensional array

参 考 文 献

1. 《PROGRAMMING IN PASCAL》
作者: PETER GROGONO
出版社: 美国 ADDISON-WESLEY
Publishing Company, INC.
出版日期: 1978年
2. 《AN INTRODUCTION TO PROGRAMMING AND PROBLEM SOLVING WITH PASCAL》
作者: G.M.SCHNEIDER
S.W.WEINGART
D.M.PERLMAN
出版社: 美国 JOHN WILEY & SONS
出版日期: 1978年
3. 《MICROCOMPUTER PROBLEM SOLVING USING PASCAL》
作者: KENNETH L. BOWLES
出版社: 美国 SPRINGER-VERLAG
出版日期: 1977年
4. 《THE PROGRAMMING LANGUAGE PASCAL》
作者: N. WIRTH
出版社: ACTA INFORMATICA
出版日期: Vol. 1, 1971
5. 《PASCAL 程序设计语言》
译者: 俞嘉惠
出版: 电子计算机参考资料1976年第1, 2期
6. 《数据结构》
作者: 严蔚敏, 沈佩娟
出版社: 国防工业出版社
出版日期: 1981年

此外, 参考了美国麻省理工学院李凡教授在协助筹建 PDP-11/03 微型计算机实验室时附来的使用手册和参考资料。